University of Southampton

# A Mobile Agent Architecture to Support Distributed Resource Information Management

by
Jonathan Dale

A thesis submitted to transfer from
Master of Philosophy to Doctor of Philosophy

in the
Faculty of Engineering
Department of Electronics and Computer Science

23 June, 1998

University of Southampton

ABSTRACT

Faculty of Engineering

Department of Electronics and Computer Science
Master of Philosophy/Doctor of Philosophy

A Mobile Agent Architecture to Support Distributed Resource Information Management

by Jonathan Dale

The problems of information management on a large and distributed scale have become highly apparent during the last five years. The information explosion that has taken place, particularly with the exponential growth of the World Wide Web, has shown that users treat information, especially hyperlinked information, as a valuable resource. However, the electronic information community is already showing signs of what has been commonly termed 'information overload'. This trend will continue to increase unless users are equipped with the necessary tools to allow them to mine and navigate this commodity.

The current set of tools that are available to the user, whilst useful, have proven to be of limited use, due to the fact that they only individually deal with relatively small subsets of the global information available.

This thesis advocates the integration of two technologies; distributed information management to provide tools to allow the user to discover, navigate and control information and mobile agent technology to allow the user to access and manage information that is distributed over large areas.

It also presents a framework within which mobile agents can move across distributed environments, integrate with local resources and other mobile agents, and communicate the results of their activities back to the user. This framework can then be used to build mobile agents that perform user-driven tasks to fulfil distributed information management goals.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

"In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move" — Douglas Adams

## 1.1 A New Electronic Information Age

There is a contemporary style of science fiction writing that has evolved over the past ten years which is known as *cyberpunk*. This form of writing differs from normal science fiction in two crucial areas. Firstly, it is directly related to the cultural concepts that have developed during the computer revolution of the late 1980s and early 1990s. And secondly, it makes a fundamental distinction about information over technology. In cyberpunk novels, electronic information is the motivating and dominating factor in the world; technology is assumed and is merely an enabling tool.

Cyberpunk writers, such as William Gibson and Neal Stephenson have both used differing terms to describe what is essentially the same concept. In *Neuromancer* (Gibson, 1984) what Gibson calls *Cyberspace* and in *Snow Crash* (Stephenson, 1994) what Stephenson calls the *Metaverse*, superficially characterise an electronic environment that is accessed through immersive virtual reality.

However, both of these terms have a much deeper meaning and more wide-ranging implications: they are both used to represent the *entire* sum of human information available in an electronic format. This information is accessible to users through powerful artificial intelligence software that can search and cross-reference information of any format very quickly. Indeed, Stephenson devotes a number of chapters to dialogue between a user and an AI program known as the *Librarian* daemon. This user has an interactive, real-time, natural English discussion with the Librarian, who collects, correlates and presents information to the user at any level of detail. The information that is available to the Librarian, and hence the user, is historical (through archives), contemporary (through current media input) and real-time (through a global network of satellites and sensors).

Whilst the realisation of such an advanced system is not dealt with at a detailed technological level, it is easy to see how the atomic elements of computer science today (for example, artificial intelligence, information retrieval and high-bandwidth networking) have been interwoven to construct a fictitious, but potentially powerful and useful electronic information environment.

## 1.2 The Reality of the Current Electronic Information Age

The current electronic information age is going through a radical change. With the increase during the past five years of inter-networked environments, such as the Internet, more electronic information is available than ever before. Computer information, which was primarily text-based until a few years ago, is also expanding into the areas of two- and three-dimensional graphics, sound, video and real-time video. The concept of a global, multimedia information space is starting to take form.

However, computers and networks of information present humanity with a fundamental problem: how is information to be organised, managed, navigated and searched on a *global* scale? These problems of distributed information management have only ever been considered on small scales previously, for example, libraries, encyclopedias and dictionaries, which for the most part had to be painstakingly compiled and cross-referenced by hand.

As more users join the electronic information age, so the human electronic information space grows. However, the problem is that it is growing in unstructured and unpredictable ways. Therefore, as it grows in an anarchistic manner, so the user finds it increasingly difficult to manage and to navigate and to obtain the information that they require.

Current technology in this area provides relatively few and generally unsophisticated tools to help the user either search for the information that they require, to manage their own sets of distributed information or to integrate with other information resources. Tools that do exist, tend to be aimed at a specific information resource and, currently, very much within the area of information discovery, for example, search engines for the World-Wide Web.

## 1.3 The Challenge for Distributed Information Management

It is obvious that a mechanism is required that is co-ordinated and developed at a global level that takes into account all current technology, but is flexible enough to be able to also incorporate future advances. However, is this possible or even desirable? How long would such a system take to design, implement and prime? What happens to all of the existing electronic information that currently exists? And, what occurs in the mean time?

Therefore, the immediate and current challenge for a global initiative for managing distributed information must be based upon current and existing technologies, must be available now and must also provide the user with flexible and expandible tools that they can customise to fit their requirements.

## 1.4 Agents of Change

Much speculative debate has been generated over the world *agent*. To some an agent is a piece of software that performs a task on behalf of a user. To others it is diverse pieces of software that collaborate to perform a task on behalf of a user. To others still, this piece of software (or pieces of software) must achieve more than simple task completion; they must possess human and mental characteristics, for example, intention, intelligence, autonomy and believability.

The essential difficulty in classifying the role or nature of an agent is difficult because they are generally ascribed tasks that would normally be carried out by a human. It is this quality that makes them difficult to define, but gives them a distinction over other pieces of software. Chapter 3 details the common characteristics that are attributed to agents and discusses their implications.

The introduction of agents represent a fundamental change in the way users will interact with their information; the emphasis will be moved away from a user interactively working with a piece of software to achieve a task to a user delegating the completion of a task to an agent. This implies that agents do have a useful and important role to play within a distributed information management system.

Additionally, agents can overcome boundaries that are difficult for normal software to cross, for example, platform heterogeneity, integration with legacy systems, movement across networks, etc. The ability for agents to adapt themselves to their environment and given situations would make them a powerful and flexible tool for users.

Assuming that the agent-user trust relationship can be developed and preserved, this means that users will benefit from improved management of their distributed information, improved navigation of other distributed information resources and improved retrieval of the actual information that they require.

## 1.5 Overview

The next chapter of this thesis examines the current distributed information management tools that are available and in common use. It continues by proposing that the concepts of discovery, navigation, consistency and integration are key and crucial aspects that any distributed information management system must posses in order to fulfil the requirements of its users.

Chapter 3 discusses the concept of an agent and how this can be related to a piece of software. Notions of agency are presented that are useful in describing the nature and boundaries of agents to ensure that they can fulfil their tasks in a sensible and predictable manner. The chapter goes on to give a taxonomy of the current perceptions of agents by the differing communities that are using them.

Chapter 4 discusses how a meta-characteristic of agents, namely that of mobility, can be employed to extend the functionality and boundaries of domains within which an agent can work. It details how mobile agents are a direct evolution from the client/server paradigm by moving the processing closer to the resource. The characteristics that mobile agent systems embody are described and the state of the art in mobile agent technologies are then presented and compared against these characteristics.

Chapter 5 describes how an architecture can be developed to support mobile agents and distributed information management. The requirements of such a system are given and then the details of a framework are outlined. This chapter hypothesises that a basic agent infrastructure must exist before integration with distributed resource systems can be accomplished.

Finally, chapter 6 outlines the future directions of this research into a prototype to realise some of the goals identified earlier. Chapter 7 presents a final overview of the research that has been undertaken and presents a summary of the conclusions that have been drawn.

# 2 Distributed Information Systems

"A distributed system is one in which I cannot get something done because a machine I've never heard of is down" — Leslie Lamport

## 2.1 The Nature of Distribution of Resources

The general trend of down-sizing that has taken place over the past fifteen years has had a significant impact on the way electronic information is organised and manipulated. Services, processing and data have moved from the traditional central mainframe of the 1970s into intranetworks of organisation-wide workstations and personal computers. Information is arranged, accessed and processed at a level that is more local to the user.

Therefore, it can be argued that the concept of distribution has been an inevitable evolution in electronic information development; information and user bases are growing at such a rate that centralised storage and processing is no longer feasible. However, what has taken place is the creation of smaller, centralised islands which provide local services and are connected to other service providers through networks and networking protocols.

A number of technologies have been developed over the years, especially for the Internet, to address the storage and access issues associated with distributed information. These distributed information systems attempt to coordinate information (called *resources*) at a server or site level and to provide mechanisms to allow the user to browse and manipulate this information.

The use of the term *distributed* in the context of networked resources, however, implies distributed access to information and possibly the physical distribution of data across a number of machines, rather than the distribution of computation or processing.

## 2.2 Distributed Information Systems

Historically, the protocols that were initially developed during the mid 1970s attempted to provide access to sets of collective information across a network. For example, the File Transfer Protocol (FTP) (Postel et al., 1985), allows users to transfer files across the network from remote machines to their own. However, this can only be achieved once the Internet address of the FTP server which hosts the file has been determined and the position of the file in the server's file hierarchy located. For large sites, the latter can be a non-trivial task.

More recently, however, protocols and systems are being developed not only to provide network access to clusters of files, such as that illustrated above, but also to describe some form of relationship between those files. These systems can then accept queries which are performed upon this relationship structure to provide potentially useful information about the nature of the files contained within the system.

To assist the user in navigating and locating the information that they need across these islands of information, tools have been built around these protocols and systems. The following two sections describe the most common tools currently in use within the Internet, arguably the largest inter-networked on-line electronic information system.

## 2.3 Distributed Resource Locators

Distributed resource locators, as their name implies, provide the user with tools to locate and subsequently access files that are shared across machines within an inter-network. Generally, they make accesses in a specific protocol to some form of distributed resource server which attempts to fulfil or resolve their actions, either personally or by contacting other distributed resource servers.

The majority of these systems are based around the client/server model, which allows clients (users) located around the world to make requests of the services that distributed resource servers provide. This affords both the client and the server a great deal of versatility, since they only need to communicate in the same protocol and in the same data format to be understood.

### 2.3.1 Archie

As mentioned earlier, one of the problems associated with FTP is that before a file can be retrieved, a user must know the Internet address of the FTP server on which it resides. When the Internet was in its earliest stages and was relatively small, this was quite successful. However, as the Internet has grown the number of FTP servers has increased dramatically.

Archie (Emtage et al., 1992) is a database service that indexes files which are available on publicly accessible FTP servers. In this way, if a user is sure of the file that they want, but is not sure of the FTP server upon which it is hosted, they can issue a query to an Archie server which returns a list of results that matched the query.

The Archie system provides a much needed service that is lacking within the FTP protocol, but can not be considered as a truly distributed information system. This is due to the fact that at periodic intervals, each Archie server must re-index its file list with each individual FTP server to ensure that it is up-to-date.

Another system that makes use of FTP is called Alex (Cate, 1992). Alex is a system for coordinating a number of FTP servers together to provide a unified FTP-space, derived from the individual file hierarchies of each FTP server. This has the advantage of providing a much larger virtual FTP server in which requests to other FTP servers are handled transparently.

### 2.3.2 Wide Area Information Server

Wide Area Information Server (WAIS) (Kahle et al., 1991) is a searching tool that allows users to specify a search criteria and then apply this to a set of selected resources. It is then the task of the WAIS server to pass the query on to each WAIS server supporting a particular resource. These servers perform a full text search against the query and return a list of matches, ranked in order of relevance.

Although the majority of resources that are available through WAIS servers are of a textual nature, since it is far easier to index, graphical images are being made accessible through the use of simple keyword association.

### 2.3.3 Gopher

The Gopher system (Anklesaria et al., 1993) is based around the concept of a distributed filing system that forms a hierarchical tree in which individual Gopher servers can incorporate information; intermediate nodes are equivalent to directories and leaf nodes are documents that may be rendered by Gopher clients.

At a first glance, Gopher may be regarded as similar in nature to FTP. However, unlike FTP, the Gopher protocol allows Gopher servers to be transparently integrated to provide a seamless view on a distributed document space, the sum of which is known as *Gopherspace*. This form of integration allows a Gopher server to place a pointer to another Gopher server at any point within its hierarchy, thus forming a primitive logical hierarchy. Additionally, Gopher search servers can be used to provide a virtual node that is the result of some search that has been performed over some part of Gopherspace.

Due to its interactive and browsing nature, there is a varied selection of information which is presently available across Gopher servers, from text to digital movies. A new version of the Gopher protocol, called Gopher+ (Anklesaria et al., 1993b), has been proposed which will enhance its capabilities in handling multimedia data, but the system has become somewhat eclipsed by the World Wide Web.

## 2.4 Distributed Hypermedia

Essentially, hypermedia is the concept of managing information in a structured and associative manner. Early hypermedia pioneers, such as Vannevar Bush (Bush, 1945), were among the first to consider using a machine (what was to become the latter day computer) as the medium through which to store, link and access this information. Interestingly enough, Bush predicted the information explosion that would occur with electronic information as far back as 1945!

The first hypermedia systems to become available (for example, Guide (Owl, 1987) and Hypercard (Apple, 1987)) were separate applications in their own right and required that all data be specially formatted and committed to the central management of the system. This approach presented two distinct disadvantages; the problem of converting large volumes of information and the problem of reusing that information outside of the hypermedia system.

These, and other limitations of early hypermedia systems, have led recent research into ‘open’ hypermedia, that is, a hypermedia system which is able to use data in its native form, without having to embed additional information which will affect its use outside of the hypermedia system's environment. Open hypermedia also advocates integration with other applications on the desktop and an open linking model, where multiple and different views can be placed upon a collection of data simply by altering the linking structure. A discourse of open hypermedia principles, motivations and potential advantages is given by Davis (Davis et al., 1992), Grønbæk (Grønbæk et al., 1992) and Pearl (Pearl, 1989).

As hypermedia systems increase in complexity and attempt to integrate with more existing systems, they have come under the same pressures as early centralised computing systems: to become distributed by expanding hypermedia data and functionality into and across networks. Distributed hypermedia systems can offer users greater flexibility in the data that they can access and share, alternative views on distributed information resources through the implementation of their own linking structures and increased heterogeneity across platforms and network architectures. The general advantages of distributed hypermedia are outlined by Goose and Dale (Goose et al., 1996).

The following sections give a brief survey of a cross-section of distributed open hypermedia systems that are in current use.

### 2.4.1 World Wide Web

Perhaps the most used and well-know distributed hypermedia system in existence today is the World Wide Web, also know as the Web, WWW and $W^3$ (Berners-Lee et al., 1992). The Web was originally developed at the international organisation CERN in Switzerland for its high energy physics community, but its applicability in a wider context soon became apparent.

The Web is essentially a client/server model where Web servers offer and perform services on behalf of Web clients. The basic element of information that servers can deal with is a document which is stored in the Hypertext Mark-up Language (HTML) (Berners-Lee et al., 1995). HTML is a subset of the Standardised General Mark-up Language that is oriented toward describing links and general layout within a document. Web clients are able to access servers and place requests through a communication protocol called the Hypertext Transfer Protocol (HTTP) (Berners-Lee, 1995b). It is the task of the Web client to parse HTML documents, to render their contents and to present any links that are present for the user. The HTML format also allows non-textual information, such as bitmaps, to be included within a document.

Documents are specified on Web servers through the use of a Uniform Resource Locator (URL). A URL consists of three components; the communication protocol to use, the Internet address of the Web server and a directory path representing the location of the document within the file hierarchy of the Web server. URLs are flexible enough to allow a number of protocols to be used, for example, FTP, Gopher, Network News (NNTP), etc.

The functionality of the Web can be considered as both an open and closed hypermedia system. It is open in its support for networking protocols, data formats and client extensibility, but there are two main criticisms of the Web which can characterise it as a closed system.

Firstly, the rigid client/server model it supports makes it difficult to scale due to potentially high pressures on servers and hinders information from being presented in its most natural form, a notable example being temporal media. However, extensions to the Web are available (such as client-pull and server-push) to help alleviate this situation, but most are non-standard. Also, Web users generally make extensive use of caches and proxies to reduce server loads and improve access times.

Secondly, the Web supports a very simplistic and weak linking model; it requires documents to be augmented with HTML mark-up, only supports one-way direct links and has a rigid linking structure. A key component of open hypermedia systems is that they can use documents in their existing format and can impose multiple views on information by changing the linking structure. Most open hypermedia systems achieve this flexibility through external link databases, which contain all of the links relevant to a particular set of documents. Research, most notably the Distributed Link Service (Carr, 1995), is being undertaken to try and apply this technology to the Web. Additionally, the Web only supports single-directional button links. This has the implication that all links of a global nature, for example dictionary entries, need to be either automatically created as a document is rendered or added by hand.

The most influential development of the Web has been the introduction of Hot Java and the Java programming language (Gosling et al., 1995). Applets are executable code that can be embedded within HTML documents, transferred across the network and executed on a variety of platforms, in an attempt to move processing from the server and onto the client. They achieve heterogeneity by being written in Java, an object-oriented programming language that is compiled to platform-independent byte-code and then interpreted by a Java-aware Web client. The application for applet code has yet to be fully realised, since current examples are quite primitive. However, it is a significant step forward for Web technology and will present some interesting developments for the future.

### 2.4.2 Hyper-G

Hyper-G (Kappe et al., 1993) is a distributed hypermedia project that is being developed at the Technical University of Graz in Austria whose original aims were to provide a general-purpose university information system to support a wide range of activities.

In a similar manner to the Web, Hyper-G functionality is provided by a set of Hyper-G servers. However, unlike the Web, Hyper-G provides a unified view on distributed resources by allowing aggregations of documents, called *collections*, to span multiple Hyper-G servers. Moreover, Hyper-G permits a more flexible linking model due to the fact that links are stored in external link databases. Although Hyper-G supports bi-directional links, they are still of a button-oriented nature, which makes global and repetitive linking difficult.

Hyper-G has been extended from its original aims to provide a large-scale, distributed, hypermedia information system that gives the intuitiveness of a top-down hierarchical navigation model with the immediacy of associative hyperlinks. Additionally, Hyper-G advocates the use of an object-oriented database to provide information structuring and link maintenance facilities.

In a distributed context, Hyper-G attempts to maintain scalability by using a document naming system that allows local replication and caching, and the integrity of link databases across server boundaries is achieved through the use of a scalable flooding algorithm. The Hyper-G model also supports a comprehensive user model which can assign permissions at the document, document group, individual user and user group level. Unfortunately, there is no provision in Hyper-G for making a distinction between a user and an author. The multi-protocol facilities of the Hyper-G system allow it to communicate with any open protocol system. Current integration is provided for native Hyper-G client, Gopher and the Web (Andrews et al., 1995), and future plans are for WAIS and FTP.

The ability to integrate with other distributed information systems makes Hyper-G a powerful hypermedia tool. Indeed, it has been suggested that all existing Web servers could be replaced with Hyper-G servers without major problems (Flohr, 1995). Hyper-G's main weakness appears to be its lack of extensibility due to a non-modular architecture, which can make it difficult to customise to user requirements.

### 2.4.3 Microcosm: The Next Generation

Microcosm: The Next Generation (MCMTNG) (Goose et al., 1995) is a distributed open hypermedia system that is based around the philosophy and framework of the Microcosm open hypermedia system. Both systems have been developed at the Multimedia Research Laboratory within the University of Southampton.

Unlike both the Web and Hyper-G, the functionality of the MCMTNG system is represented by a set of asynchronous, communicating processes. This modular, process-driven architecture has the advantage that processes can be added to the system to increase or augment its functionality, or to customise it to a particular user's requirements. What is different about the MCMTNG system is that it exists at the user-level, rather than at the site or domain level; there can be multiple MCMTNG systems executing within a domain for a number of given users simultaneously.

Local hypermedia functionality is handled by the processes of a user's MCMTNG system. These processes, primarily link databases, document managers, viewers and link managers, are all registered with and attached to a local message router; they register their interests with the router, send messages to the router and receive relevant messages from the router. The hypermedia model is inherited from the Microcosm open hypermedia system (Davis et al., 1992; Hill et al., 1993) and provides for external link databases and multiple link types. The most important of these is the generic link; a generic link is authored only once and automatically applies to all selections which match the original link creation data.

When MCMTNG systems inside or outside of a domain wish to communicate, they initially make contact through a domain server. The domain server provides information about the MCMTNG systems that are executing within the domain and the information resources that are available. Once the communicating parties have been established, communication subsequently occurs through the two message routers of the respective MCMTNG systems.

The MCMTNG system extends the original concept of a hypermedia application from Microcosm and re-fashions it in a distributed context. A hypermedia application is now a reflexive entity that can contain documents and/or pointers to other hypermedia applications, similar to a Hyper-G collection. This flexibility allows users to create large hypermedia structures from individual hypermedia applications, according to their interests. Indeed, users can place their own views on hypermedia applications by rearranging, removing or adding other hypermedia applications.

The MCMTNG system is currently in a prototype stage. However, the flexible and extensible nature of the architecture and the use of open protocols mean that it is possible to integrate with other systems and protocols. The future work of the MCMTNG system lies in exploring the possibilities of the reflexive hypermedia application framework within the context of a widely distributed environment.

## 2.5 Distributed Information Management

Distributed information management is a term that has been traditionally been used to describe the integration and management of distributed database systems. However, more recently, it has been used to describe the integration and management of distributed information systems and resources across networks and protocols (De Roure, 1996).

The following sections describe four issues which are key to achieving successful distributed information management.

### 2.5.1 Resource Discovery

The purpose of resource discovery is to search through distributed information systems and to present new sources of relevant information to the user. Since resource discovery mechanisms are being employed due to the fact that there are too many distributed information systems for the user to search manually, the searching algorithm must be accurate to ensure that relevant data is not overlooked and data that is not relevant is discarded before it reaches the user.

Another function of resource discovery is resource monitoring: the active task of notifying the user when the contents of a resource change. This is particularly useful if the user is monitoring temporal media, for example stock prices, but can also be used to indicate when a user should revisit information resources.

Fortunately, nearly all of the distributed information systems described previously contain some form of searching algorithm that allows their textual contents to be interrogated. Unfortunately, some provided richer search mechanisms than others. For example, FTP provides a very weak searching mechanism since it indexes on file name, whereas Web search engines index on document contents.

### 2.5.2 Information Integrity

As information becomes distributed across wide areas, information integrity becomes a real need and a real problem. Due to the problems of packet loss and latency associated with networks, it is difficult to ensure that consistency updates are made in a timely fashion. Additionally, when considering collaborative working environments, versioning and update control needs to be implemented to ensure that edits are not lost.

In terms of hypermedia systems, the problems of link and document consistency must also to be handled. Link consistency deals with ensuring that the integrity of links is maintained, even if the source and/or destination anchor moves. In most cases, link inconsistency is relatively easy to deal with, since it either involves removing the link (if it is no longer valid), or re-pointing the start or end anchor to the new location.

Document consistency is a much more difficult problem, since it deals with the contents of a document changing and can also infer link inconsistency. If the end anchor of a link points to a keyword in the centre of a given document and that keyword is subsequently deleted in an edit, how is this resolved?

Unfortunately, in most instances of inconsistency, some form of user intervention will be required to ensure that the damage is repaired correctly. In all but the most trivial of cases, consistency algorithms can do little more than highlight the problem for the attention of the user. However, in distributed hypermedia systems that employ separate linkbases, such as Hyper-G and MCMTNG, the task of consistency checking is made simpler since interaction only occurs between the linkbases. Where link information is embedded, as with the Web, interaction must occur between all documents within the system.

### 2.5.3 Navigation Assistance

Navigation assistance is a process of assisting the user in navigating some form of information resource. This information resource could be the information contained within a distributed information system or could be the information generated by a number of resource discovery algorithms. Either way, a navigational assistance algorithm can be employed to protect the user from information overload.

Oren (Oren, 1987) sums up the role of this algorithm as:

> "A parallel would be the human reference librarian who does not comprehend the material in articles being sought, but does understand the conventions of card catalogues, abstract collections, citation indexes and bibliographical references. Because these relations can be made explicitly in hypertext they can be utilised without, for instance, having any deep comprehension of the meaning of any article title."

Wilkins (Wilkins, 1994) further describes navigation assistance as an algorithm that can fulfil the following requirements:

- Reduce the number of possible destinations for a user to choose from when too many are offered.
- Expand the number of destinations for a user to choose from if none are apparent.
- Gather and intelligently combine existing sources of information (for example, navigation) that already exist within the system and present the user with one combined source of information instead of many.

### 2.5.4 System Integration

A key aspect of distributed information management is the ability to manage information resources that are on heterogeneous networks, heterogeneous platforms and are represented by heterogeneous protocol formats. To this end, there is a need for distributed information management tools to be able to integrate with a wide range of distributed information systems

However, integration is more than protocol conversion, since there is a semantic problem to be overcome. For example, how are links translated between the Web and MCMTNG? If there is more information represented in a MCMTNG link, how is this stored within a Web link? Further more, is it possible to apply links *across* distributed information systems? If so, where are these links stored and who resolves them?

This illustrates that there are two fundamental approaches to system integration: arming distributed information management tools with the necessary information to be able to converse with multiple distributed information systems and equipping them with the necessary information to make semantic protocol conversions.

## 2.6 Summary

This chapter has highlighted the essential problem that faces all inter-network users today, and possibly for the future. As systems such as the Internet continue to grow unbounded, then the amount of information that is available within the system will also continue to grow. This leads to the concept of information overload.

Currently, each distributed information system possesses its own set of tools to assist the user in navigating its information resource. However, these tools are only of value within that particular distributed information system and generally only perform the function of resource discovery. Therefore, there is a real need to develop tools that cross distributed information system boundaries to provide the user with a view on the entire collection of information resources that are available.

Additionally, distributed information management shows that the user requires more tools than simplistic resource discovery mechanisms. They require additional discovery aids, navigation aids and maintenance aids, each of which should integrate with a wide range of distributed information systems and should also perform protocol conversion (both syntactic and semantic) in a sensible fashion.

This thesis advocates that a potential solution to this problem is to employ agent technology to perform these task on behalf of the user. These agents would be given goals by the user and work autonomously and intelligently to achieve those goals. The next chapter examines the state of the art in agent technology and attempts to classify the differing views on what an agent actually represents.

# 3 Agents

"**agent** (a'jent) n. a person who acts on behalf of another person, business or government, etc. [C15th: From Latin *agent-*, noun use of the present participle of *agere* to do]" — The Collins English Dictionary

## 3.1 What *is* an Agent?

The problem of giving a complete description of an agent is difficult, due to the wide range of tasks that an agent can perform and the general perceptions of the user community. However, in its broadest sense, all of the following scenarios could be attributed to the work of an 'agent' (adapted from (Wooldridge et al., 1995)):

- Upon logging into your computer system, you are greeted by your personal digital assistant (PDA), who presents you with a summation of activities that have occurred since you last logged in. The PDA has sorted your email messages into order of importance and has automatically cross-referenced them to find any related email messages. One email in particular is from your supervisor requesting a meeting to discuss your work; the assistant has checked your calendar and already negotiated a suitable time and date with the PDA of your supervisor. In addition to this, your PDA has also searched the network news groups and presents you with a list of articles that it knows you are interested in; the assistant draws your attention to a thread which describes work that is very closely related to your own. In anticipation that this will be of interest to you, the PDA has already obtained a technical report from an FTP site and compiled a list of other references which are available and detail this work further.

- You are editing a file when your PDA requests your attention; an email message has arrived that contains notification of acceptance of a paper that you submitted to a conference. Without prompting, your PDA has looked into travel

> arrangements and has determined all of the planning requirements, from taxis to aeroplanes. In a short while, you are presented with a summary of the most convenient travel options. Once you have confirmed a suitable option, your PDA makes reservations on your behalf, marks your calendar accordingly and negotiates with the conference site about locating your data locally for the duration of the conference. While you are in transit to the conference, the assistant transfers your data (including the files necessary to your presentation) to the conference site, redirects your information sources (email, for example) and informs the PDA of the conference supervisor of your impending arrival.

Although some of the technology to support computer systems of this level of sophistication is not available, research is being undertaken in these areas. However, the key computer-based element that occurs in each of the above scenarios is termed an *agent*.

At an elementary and conceptual level, agents can be regarded as entities that perform actions on a person's behalf. For example, we visit a travel agent and book our holidays; the travel agent is given the task of organising the flight details, the hotel bookings and the insurance arrangements on our (the holiday maker's) behalf. We do this because it would be tiresome to have to organise all of the details ourselves; we delegate responsibility to an agency we know can perform the task.

An analogous description in the software world could be that an agent is 'some software that performs tasks on a user's behalf'. However, in blithely making this parallel and believing that the analogy translates correctly, a number of fundamental assumptions are being taken for granted. In our real-world scenario, we *trust* a travel agent to book the flight and we *believe* that they will book the hotel for the correct dates. In other words, we have delegated not only our trust in the ability of the travel agent to complete the task that we have conferred to them, but we also believe that they will execute it correctly.

Therefore, the usefulness of an agent is directly related to the amount of trust and believability it generates and this is possibly more important than the actual task it can perform; agents that fulfil their tasks badly (either incorrectly or incompletely) are of little use to us, since we would employ the services of another agent or simply complete the task ourselves, rather than use it again.

These concepts of trust and believability need to be identified and embodied in software agents (hereafter referred to simply as an *agent*), before their tasks can be accorded the same amount of trust that is granted to real-life agents.

## 3.2 Notions of Agency

*Agency* is concerned with the concepts and attributes that can be assigned to agents to determine their nature and to predict their behaviour. An agent whose nature is well defined and whose behaviour is predictable is more likely to be of use and to be trusted by the user. Wooldridge and Jennings (Wooldridge et al., 1995) detail two notions of agency, weak and strong, that are useful when describing the nature and actions of agents.

### 3.2.1 A Weak Notion for Describing Agents

An agent is considered to adhere to a weak notion of agency if it is comprised of the following properties:

- *Autonomy*. Once launched with the information describing the bounds and limitations of their tasks, an agent should be able to operate independently from their user, that is, autonomously in the background (Castelfranchi, 1995). To this end, an agent needs to have control over its actions so that it can determine what to do when an action succeeds or fails. Moreover, an agent must be able to augment its internal state so that it can make rational decisions based upon the information that it has gathered.

- *Social ability*. To effect changes or interrogate their environment, an agent must possess the ability to communicate with the outside world (Genesereth et al., 1994; Mayfield et al., 1995). This interaction can exist at a number of levels depending upon the remit of the agent, but typically an agent would need to communicate with other agents and the local environment (to maintain/discover information) and users (to appraise them of their progress).

- *Reactivity*. Agents need to be able to perceive their environment and respond to changes to it in a timely fashion, depending upon their remit. For example, an agent's task could be to monitor a local file system, informing the user when changes occur to a particular file set. This implies that the agent has an awareness of the appropriate filing system and how to interrogate it; agents need not only to be aware of their environment, but they need to be aware of what the state and changes in that environment mean and how to react to them.

- *Pro-activeness*. To help differentiate an agent from another piece of software, agents need to be able to exhibit pro-activeness, that is, the ability to effect actions to achieve their goals by taking the initiative. This means that an agent needs to appreciate the state of their environment and to decide how best to fulfil their mission target.

When using these attributes to describe agents, it is often referred to as agent-based software engineering (Genesereth et al., 1994). This programming model supports the concept of an agent as a self-contained, concurrently executing process that contains and controls some internal state and accesses its environment and other agents through a message passing protocol.

### 3.2.2 A Stronger Notion for Describing Agents

Further notions of attributes for agents include descriptions that possess more specific meaning than weak agency, that is, they are attributed characteristics and tasks that would normally be ascribed to humans. Shoham (Shoham, 1993) describes the mentalistic notions of knowledge, belief, intention and obligation that might be attributed to strong agents above and beyond those defined for a weak agent. Dennett (Dennett, 1987) describes such an agent as an *intentional system*.

An intentional system is a system that can best be described by the *intentional stance*; the ascription of abstraction notions to systems for the purpose of describing how they work. For example, although the technical description of a computer system may be available, it is too complex to use when describing, say, why a menu appears when a mouse button is clicked over a certain area of the display. The intentional notions as described by Shoham are useful for providing convenient and familiar ways of describing, explaining and predicting the behaviour of complex systems: Dennett suggests that strong agents are best described by the intentional stance.

Bates uses this the concept of strong agents and takes it into more anthropomorphic areas by considering the implications of believable agents, that is, agents that try to model a human approach to their interaction with the user by displaying emotions (Bates, 1994). Additionally, Maes talks about representing agents visually by attaching an icon or a face to associate them with cartoon or computer characters (Maes, 1994). These types of agents are being used in both Human Computer Interaction (HCI) scenarios to help the social interaction between a user and their agents, and also in the computer gaming community to produce virtual characters that react in believable and human ways to given situations.

### 3.2.3 Intelligence and Learning

Intelligence is an attribute that can be given to both weak and strong agents and determines how agents will react to situations and events. In most agent communities, intelligence is seen as the key factor that separates agents from ordinary pieces of software.

However, as Brooks stated (Brooks, 1990; Brooks, 1991; Brooks, 1991b), intelligence is 'in the eye of the beholder', since the question 'what is intelligence?' is as difficult to define as 'what is an agent?'! He goes on to make two key points about intelligence:

- Real intelligence is situated in the world, not in theorem provers and expert systems, and,
- Intelligent behaviour arises as a result of an agent's interaction with its environment.

The problems of artificial intelligence seem to lie in two areas (Shardlow, 1990); the translation of the real world into an adequate description in time for it to be useful (*transduction*), and, the representation of complex systems and entities and how to reason about this information in time for it to be useful (*reasoning*).

Edwards (Edwards, 1995) argues that it is not enough for agents to react intelligently to their environment, but they must also be able to adapt and alter to changes by *learning*. An agent that can learn through exposure to given situations and examples could be more useful to a user than an agent whose intelligence is fixed. However, it is far more difficult to predict the behaviour of an agent that can learn, since it is not possible to determine exactly what it will learn and how it will apply that information.

### 3.2.4 Other Attributes of Agents

A number of other attributes can be given to both weak and strong agents to augment or temper their functionality. These include, but are not limited to (Wooldridge et al., 1995):

- *Mobility*. The ability for an agent to move across networks to fulfil its goals (Gray, 1995b). Mobility will be explored in greater depth in the next chapter, but it is enough to say here that it introduces some interesting concepts of persistence and 'lifetime'.

- *Rationality*. The assumption that an agent will not act in a manner that prevents it from achieving its goals and will always attempt to fulfil those goals (Galliers, 1988).
- *Veracity*. The concept that an agent will not knowingly communicate false information (Galliers, 1988).
- *Benevolence*. An agent cannot have conflicting goals that either force it to transmit false information or to effect actions that cause its goals to be unfulfilled or impeded (Rosenschien et al., 1985).

Goodwin formally defines these and other agent attributes using the Z formal specification method (Goowdin, 1993).

### 3.2.5 Summary

Agent classifications are useful in illustrating some of the ideas that lie behind the development of agents for both theoretical and real-world applications. Whether embodying weak or strong notions of agency, an agent needs to act in a manner that forges a trust relationship with the user. If an agent cannot complete a task, the user must be informed of what when wrong and why. Similarly, if the agent successfully completes a task, then the user must be informed of the details of how this was accomplished and the results generated, so that they can be verified.

Intelligence and learning appear to offer methods for building trust and believability into agents; intelligence allows agents to fulfil tasks competently and learning allows agent to react to situations in an intelligent manner. To ensure that an intelligent and learning agent does not grow outside of its remit, other agent attributes, such as benevolence, veracity and rationality may need to be incorporated.

## 3.3 The Differing Views on Agents

The following is a brief taxonomy of the various perceptions that differing computer science disciplines hold about agents (Wooldridge et al., 1995).

### 3.3.1 The Traditional Agent

The traditional concept of agents began, not surprisingly, with the artificial intelligence community. It is a view based around agents being systems that can take input data about their environment, reason about it and (possibly) generate appropriate output responses (Kurzweil, 1990).

The ultimate goal of AI agents is to provide intelligence and reasoning capabilities that are comparable to that of human beings. As McCarthy (McCarthy, 1978) puts it:

> "[Artificial Intelligence is] the science of making computers do things which if done by humans would require intelligence."

However, due to the difficulties of capturing and describing the essential qualities of intelligence, the AI community have come to recognise that agents are a suitable vehicle for expressing the desirable properties of artificial intelligence. Indeed, a common theme that is shared by most AI computer scientists is that because agents possess a capacity of intelligence, this differentiates them from other, normal pieces of software.

Traditional AI architectures are generally based around three core philosophies:

- *Symbolic*. The reduction of the world to a representation of realisable symbols that can be combined to form structures upon which processes can be executed to operate upon the symbols according to symbolically coded sets of instructions (Newell et al., 1976). Decisions regarding actions to perform are made through logical reasoning, based upon pattern matching and symbolic manipulation. Example systems include Homer (Vere et al., 1990), a robot submarine which explores a two-dimensional Seaworld, can respond to an 800-word spoken vocabulary and has a limited episodic memory, and Grate* (Jennings, 1992), a simulation of electricity transportation management where agents act in an individual, selfish and cooperative manner to achieve their intentions.

- *Reactive*. A major problem with symbolic AI is the processing power required to analyse the information about the real world, plan a suitable solution and then implement a chosen action. Critics of symbolic AI have advocated the use of reactive architectures; architectures where there is no complex representation of the real world in symbolic terms and where no symbolic reasoning is performed. The most vociferous critic, Rodney Brooks, has developed an approach based around a reactive model, called the *subsumption architecture* (Brooks, 1986). A subsumption architecture is a hierarchy of behaviours designed to accomplish tasks. Each behaviour competes with other behaviours in order to influence the actions of the agent; lower layers in the hierarchy represent primitive behavioural styles (for example, avoiding obstacles in the Homer Seaworld) and higher layers represent more abstract behaviours (for example, collecting an object and returning it to a given location). The amount of computation required for these systems is very small when compared with symbolic AI systems. Another reactive architecture, based along the same lines as the subsumption architecture, is Pengi (Agre et al., 1987), a computer simulated game where routine tasks are encoded in low-level structures and are only updated to handle new problems that develop. The idea is that most decisions are routine and can therefore be performed quickly and efficiently.

- *Hybrid*. These types of architectures attempt to marry the best qualities of both symbolic and reactive approaches to AI. The method consists of building an agent system out of two (or more) subsystems; a symbolic world model which develops plans and makes decisions, and a reactive subsystem which is capable of reacting quickly to events that happen without having to resort to complex symbolic manipulation. Examples of such hybrid systems are Touring Machines (Ferguson, 1992) and InteRRaP (Müller, 1995). In these architectures a layered model is employed; a symbolic AI engine sits at the top of the model handling long-term goals and a reactive AI engine sits at the bottom handling low-level reactions. The problem with such systems is how to manage the interactions between the layers.

### 3.3.2 The Interface Agent

An interface agent is described by Maes (Maes, 1994b) as:

> "...a *personal assistant* who is collaborating with the user in the same work environment."

Thus, interface agents assist the user in whatever tasks they are performing, maybe to provide insight into specific situations or to provide alternative material into related areas of work.

To support this line of reasoning, the AI laboratory within MIT have developed a prototype interface agent called News Tailor, or NewT (Sheth, 1994; Maes, 1994). A NewT agent is a USENET news filter that can be 'trained' by giving it a series of examples that show in which kind of articles the user is interested. From this, the NewT agent can search all news articles to try and find other articles which are similar to the one's initially indicated by the user. When the agent presents the other articles that it has found, the user gives feedback according to their applicability; thus, the NewT agent can widen or restrict its searching next time.

Other interface agent systems include *NewsWeeder* (Lang, 1995), *UNA* and *LAW* (Green et al., 1995), *WebWatcher* (Armstrong et al., 1995) and *LIRA* (Balabanovic et al., 1995).

### 3.3.3 The Information Agent

An information agent is one that has access to a number of information resources and is able to collect and manipulate that information. Typically, it can communicate across the network to locate information resources to query or manipulate. An example might be where an information agent is asked to find a particular paper; the information agent searches a number of information resources and presents the user with FTP sites and WWW addresses, for example.

The key qualities of information agents lie in their ability to communicate with a wide range of information resources to ensure that the widest amount of information is processed to provide the user with the best results to their original request.

Theoretical studies on how agents can utilise the information that they receive from different resources are presented by Levy (Levy et al., 1994) and Gruber (Gruber, 1991). A more practical application has been presented by Voorhees (Voorhees, 1994), who describes a prototype system called the Information Retrieval Agent (IRA) which can search for loosely specified articles from differing document repositories.

### 3.3.4 The Distributed Agent

Distributed agents (also known as multi-agent systems) are collective agents which together sit at the macro (social) level, rather than the micro (agent) level. Distributed AI (DAI) (Bond et al., 1988) looks at how problems at a macro level can be broken down into agents at the micro level and how those agents can be made to co-operate and co-ordinate their activities to ensure that the problems are solved efficiently.

DAI agent technology is being employed in many real-world situations, for example, air traffic control (Steeb et al., 1988), particle accelerator control (Jennings et al., 1993) and telecommunications network management (Weihmayer et al., 1994). However, a key problem with DAI is ensuring that problem decomposition and subsequent communication and discussion between communities of agents can take place timely enough to produce useful and achievable results.

## 3.4 Summary

Although useful for breaking down classifications for agents, the views that are used and put forward by computer scientists are not necessarily compliant with the notions of weak and strong agency as described by Wooldridge and Jennings.

Most technologies that use agents realise that the more attributes that an agent possesses, the more complex the task becomes of specifying, designing and implementing that agent. This helps to explain why there has been a general trend over the past 10 years away from AI dreams (the HAL computer, for example, from 2001) to more realistic areas of actual applicability.

Agents that are useful to the user in everyday activities seem to be the way that agent technology as a whole is moving. It is hoped that by starting in the small with relatively easily specified agents which have limited capabilities and limited intelligence/learning, then the experience gained will begin to show the way of progressing up to computing with agents in the large.

General progress has been made over recent years, especially in the area of information agents, and this appears to be where future research is heading. The following chapter discusses a particular aspect of agent technology, mobility, and describes how applicable current mobile agent systems are for distributed information management.

# 4 Mobile Agent Systems

"One doesn't discover new lands without consenting to lose sight of the shore for a very long time" — Andre Gide

## 4.1 The Mobile Agent Paradigm

Mobile agents are a classification of agents whose predominant feature is the ability to transport between nodes on a network or between nodes across networks. They are the basis upon which true distributed information management agents can be built.

Mobile (or transportable) agents are a direct extension of the client/server technology. In the client/server paradigm (Renaud, 1993), communicating entities have fixed and well-defined roles; a *server* offers a set of services and a *client* makes use of those services. This model also implies a strict sense of dependency; clients are dependent upon servers to provide the services that they require. The communication mechanism that takes place between a client and a server is through a message passing protocol since network communication is assumed. However, message passing has been criticised as being too low level, requiring programmers to determine network addresses and synchronisation points themselves.

The Remote Procedure Call (RPC) system developed by Sun Microsystems Incorporated (Bloomer, 1992) attempts to remove the burden of these details from the programmer. This is achieved by allowing the client to request a service to be executed on a server in the same way that it would make a local function call. These services are represented by *stubs*, which are template function calls that pass through to the RPC subsystem. The location of the server, the initiation of the service and the transportation of the results are handled transparently to the client.

However, a fundamental problem exists with client/server architectures when considering distributed information management. If the server does not provide the exact service that the client requires, for example the server only provides low-level services, then the client must make a series of remote calls to obtain the end service that it requires. This may result in an overall latency increase and in intermediate information being transmitted across the network which is wasteful and inefficient, especially for large amounts of data. Moreover, if servers attempt to address this problem by introducing more specialised services, then, as the number of clients grow, the amount of services required per server becomes unfeasible to support.

Other client/server systems include the Common Object Request Broker Architecture (CORBA) (Object Management Group, 1993) which attempts to make the client/server paradigm more accessible by adopting the object-oriented principles of object reuse, inheritance and encapsulation, and, the Distributed Computing Environment RPC (Open Software Foundation, 1992) which offers security and authentication facilities and an interface of user-level threads instead of sockets to achieve a higher level of abstraction.

Subprogramming alleviates the problem of client/server architectures somewhat by allowing clients to launch subprograms at the node where the service is located. In this way, any number of requests can be initiated locally and the subprogram can process the intermediate results before transmitting the actual results to the client once it has finished. Example subprogramming systems are SUPRA-RPC (Stoyenko, 1994), Remote Evaluation (REV) (Stamos et al., 1990) and Network Command Language (NCL) (Falcone, 1987).

Although subprograms can migrate on their initial launch, they cannot subsequently move to other systems or resources. This restricts their ability to communicate with each other since they may not execute for large periods of time and also maintains the rigidity of the client/server relationship. Additionally, subprograms are generally written explicitly for a specific client, so their reusability is minimised.

The mobile agent paradigm attempts to address the issues that are raised by the client/server and subprogramming paradigms. Typical characteristics of mobile agents are their ability to migrate at will, autonomy in their actions, a peer-to-peer personality and a processing and network independence from their original location.

Mobility is a desirable characteristic in agents for a number of reasons (Harrison et al., 1995):

- *Efficiency*. If an agent can move across networks to the location where resources reside, then network traffic can be reduced since the agent can preprocess data and decide which is the most important information to transfer. This is a crucial aspect when considering users who connect through a low bandwidth link.

- *Persistence*. Once a mobile agent is launched, it should not be reliant on the system that launched it and should not be affected if that node fails. The concept of an agent moving between network nodes gives it the ability to 'survive' and to reach as many resources as possible. This is useful for mobile computer users due to the fact that they can log on, launch an agent, log off and check later on its progress.

- *Peer-to-peer communication*. A failure of the client/server paradigm is the inability of servers to communicate. Mobile agents are considered to be peer entities and, as such, can adopt whichever stance is most appropriate to their current needs. For example, when a mobile agent is interrogating a resource it takes the role of a client. However, when another mobile agent wishes to query it, then it becomes a server. This allows for great flexibility in dealing with network entities and distributed resources.

- *Fault tolerance*. In a client/server relationship, the state of the transaction is generally spread over the client and the server. In the event of a network or server failure during a request, it is difficult for the client to reclaim the situation and re-synchronise with the server because the network connection will have been lost. However, since mobile agents do not need to maintain permanent connections and their state is centralised within themselves, failures are generally easier to deal with.

## 4.2 Characteristics of Mobile Agents

Although mobile agent systems may support differing sets of functionality and can be implemented in different ways, they all possess a similar core philosophy. The following sections describe the key characteristics of mobile agents.

### 4.2.1 Migration

As has already been stated, mobility is the characteristic that allows agents to move between network nodes, but *migration* is the function which controls how this transfer is achieved. Although a mobile agent is essentially an executing process, the governing factor that distinguishes it from a normal process is the fact that not all of its instructions have to be executed on the same node.

Just as a mobile agent is fundamentally different to a process, so too is agent migration different from process migration; the difference lies in *who* decides where and when to move. In process migration, migration is normally forced upon a process by the system, due to resource location, load balancing and other similar factors. This is generally a complex and intensive operation. With mobile agents, it is *the agent* who decides when to move and the underlying infrastructure must support and execute this request. Enforced migration can only be effected upon an agent in extreme circumstances, for example, if the agent attempts to perform a forbidden action.

There are two approaches to moving agents between network nodes:

- *State-oriented*. This system allows agents to move at any point in their execution, usually through a mobility imperative (such as `go`, `jump` or `move`). When this command is initiated, the current state of the agent is encapsulated and transferred across the network to the receiving network node. Once received, execution of the agent resumes at the instruction following the mobility imperative. The capture of an agent's state can be achieved either by the agent itself or by the infrastructure, both of which are non-trivial tasks.

- *Stateless*. With a stateless system, agents can move at any point in the code, but restart their execution from the beginning of their code rather than the point of migration. This means that before the agent can move, all state information must be written to a data repository or knowledge base that is encapsulated and transferred with the agent. At the receiving network node, if it needs to, the agent must restore its state manually from the repository. Such systems are easy to migrate, but make writing the agent difficult since all of the state store/restore burden resides with the agent.

### 4.2.2 Data Acquisition

Mobile agents interrogate their local environment to acquire the information necessary to achieve their goals. This information needs to be filtered locally by the agent before it is either stored with the agent or forwarded to some receiving destination (such as the original network node of the agent). Therefore, mobile agent systems need to have an appreciation of the resources with which they are working.

### 4.2.3 Route Determination

Once an agent has finished with a network node, it must make a decision of where to move to next. The acquisition of data leading to this decision can be derived by one of three methods:

- *Predetermination*. This is where the agent is given the destinations that it must visit when it is launched. This is useful in situations where the activities of an agent need to be carefully controlled, for example, when making consistent alterations to resources or where the order in which network nodes are visited is important.

- *Dynamic determination*. This method dictates that the agent has been given complete freedom over the network nodes that it can visit. The mobile agent may simply make a random choice, which is quite difficult unless it knows what nodes are available, or it may make a decision based upon its own node-knowledge history or information gained from other agents. This type of freedom is useful for data mining agents where a wide field of interrogation is required.

- *Hybrid determination*. This is an amalgamation of pre- and dynamic determination methods. Put simply, the mobile agent is given a list of destination nodes that it must (or must not) visit, but can still exercise some discretion if a potential node offers data or services that are compatible with the agent's goals. Alternatively, the agent is equipped with a set of criteria that determine the type of nodes that it can visit; nodes that do not fit these criteria cannot be visited by the agent. This is useful in situations where the speed (and maybe quality) of data return is more important that the depth of data covered.

### 4.2.4 Communication

The ability for agents to communicate is fundamental to mobile agent systems. There are two methods for agent communication to take place:

- *Network-oriented*. Agents communicate through some network-based mechanism, such as message passing. This means that the communicating parties do not have to be residing on the same node or even the same network.

- *Node-oriented*. Agents communicate through some local inter-process communication mechanism, such as files, shared memory or anonymous pipes. This means that the communicating parties must be currently executing on the same network node.

Additionally, communication can take place in two basic forms:

- *Synchronously*. The communicating parties must arrange for a time to communicate and must synchronise before data can be transferred. This is typically used for situations where data transfer is important and needs to be confirmed or there is to be some interactive dialogue, for example, agents querying each other's knowledge bases.

- *Asynchronously*. The communicating parties can communicate with each other at will: the data is received as the receiving party checks for it. This is a much lazier approach than synchronous communication and is generally used for transferring informative data.

The parties that a mobile agent may wish to converse with include:

- *Local environment*. Mobile agents are restricted to communicating with the local network environment to ensure that the intermediate data transfer problem associated with the client/server paradigm does not arise; they must move between local environments to access data. The only time agents are allowed to communicate with remote environments is in determining the resources or services that exist there, in trying to determine the location of another agent or in transferring results back to the original node of the agent.

- *Other agents*. Communication between agents is useful in a number of contexts; to allow agents to share information about nodes, to allow agents to interrogate other agents' knowledge bases and in coordinating activities between agents for collaborative work. Nearly all current mobile agent systems provide some form of communication abstraction for agents, typically called `meet` or `talk` for example, which simplifies the process of agent location and negotiation. However, it is no trivial matter to organise meeting points between mobile agents, due to their mobility!

- *Users*. At some point, mobile agents will need to communicate their results back to the user that initiated them. Typically, mobile agents log their data with an agent at the user's network node and this either presents the information 'as is' or preprocesses it in some fashion. In most resource discovery activities, this task is often performed by another agent which collates the information and organises it into a format which is suitable for the user to view. Additionally, this agent may integrate with the user's desktop environment to present the information using the most appropriate applications, for example, rendering RTF in Word or spreadsheet information in Excel.

## 4.3 A Survey of Mobile Agent Systems

With the interest that mobile agents are receiving as a potential tool to assist the user control information overload a number of architectures have been developed which, while not directly addressing the task of distributed information management, do help to provide a framework within which such a system could be developed. The following sections detail the most prevalent systems currently available.

### 4.3.1 Agent TCL

The Agent TCL system (Gray, 1995; Gray, 1995b) is a model for supporting transportable (mobile) agents that is being developed within the Department of Computer Science at Dartmouth College. The architecture of Agent TCL is based upon the server model advocated by Telescript (White, 1994) and the initial language implementation is centered around an augmented form of the Tool Command Language (TCL) (Ousterhout, 1994). The architecture consists of four levels, as illustrated in figure 4.1.

All services that are available within the system are provided by agents, transportable or stationary. Agents can be written in a language that supports interpretation, such as TCL or Java (Gosling et al., 1995), but the authors indicate that compiled agents might be possible in a limited capacity. For example, an agent written and compiled in C might not be able to migrate, due to its platform dependence.

At each Agent TCL site, a server resides and handles the management of local agents and incoming agents. The server also provides mechanisms for enforcing security, providing a hierarchical name space in which agents can be referenced and allowing agents to address each other locally.

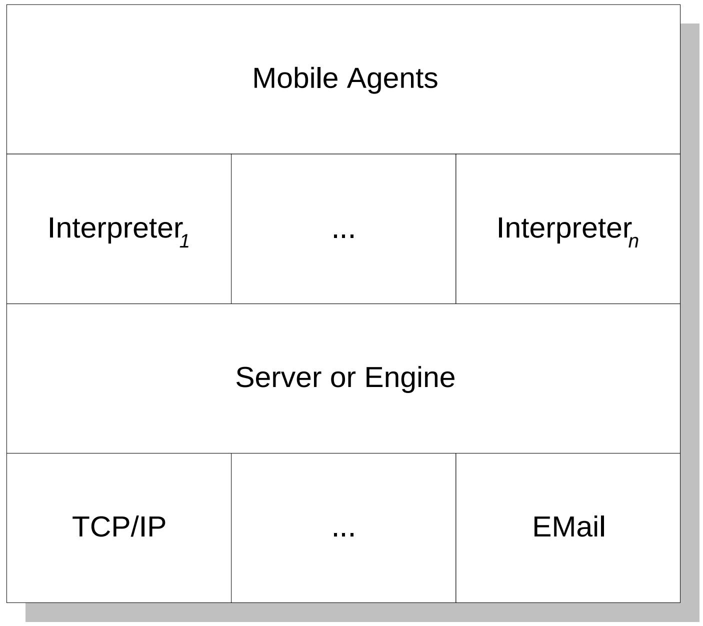


Figure 4.1: The Agent TCL Architecture

Agents move between sites in a state-oriented fashion by issuing the mobility imperative `agent_jump`. This command packages the state of the agent and transfers it to a destination site where the server restarts the agent at the instruction after the mobility imperative. The method in which the agent is communicated is determined by the transport system advocated by the local site server, for example, TCP/IP, email, etc.

The execution of agents is handled by an interpreter that is appropriate to the source language of the mobile agent. In Agent TCL, the interpreter of TCL was extended to support three extra modules (figure 4.2); a *security module* that prevents an agent from performing malicious actions, a *state capture module* that packages and restores the internal state of an agent and a *server API module* that allows interaction with the server to provide migration and agent communication, for example.

The alpha release of Agent TCL only supports synchronous, network-oriented communication through the commands `agent_meet` to initiate a communication with another agent, and `agent_accept` which completes and synchronises the two agents. Also, movement between sites is predetermined with a set of machines being specified in the agent's code. However, future work will look at making agents aware of their environment so that they can plan a migration strategy.

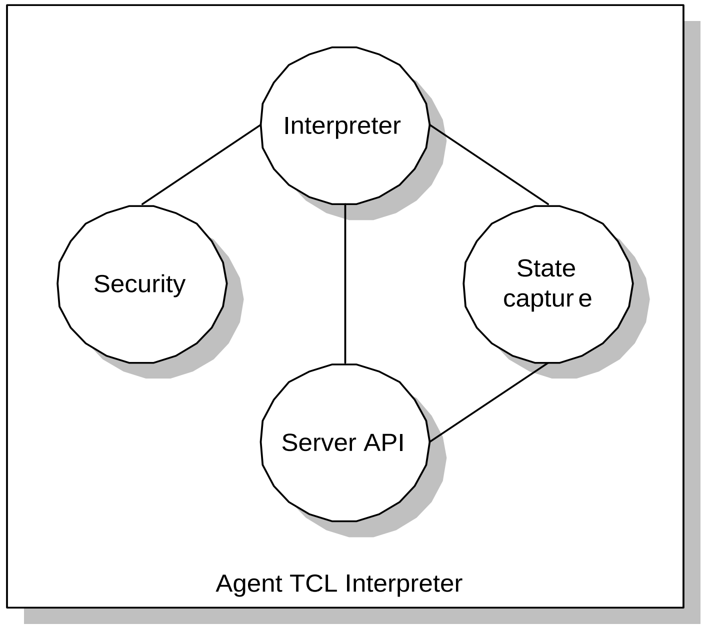


Figure 4.2: The Components of an Agent TCL Interpreter

The Agent TCL architecture has been used in three information retrieval applications. The first is in the domain of technical reports, the second in text-based medical records and the third in three-dimensional drawings of mechanical parts.

### 4.3.2 TACOMA

Tromosø And COrnell Moving Agents (TACOMA) (Johansen et al., 1995; Johansen et al., 1995b) is a joint project that is being developed by the University of Tromosø and Cornell University and is primarily concerned with providing operating system support for agents. TACOMA has been through many stages of revision, but the latest prototype uses TCL/HORUS which is a version of the TCL scripting language that uses HORUS (van Renesse et al., 1995) to provide group communication and fault tolerance.

TACOMA considers agents, either stationary or mobile, to be the computational unit of the system. As such, agents that are mobile operate in a stateless mode and each agent has three storage mechanisms. A *folder* is the essential unit of data that is accessible by an agent (figure 4.3); folders can be stored in *filing cabinets* which are stationary data repositories, or in *briefcases* which are containers that agents carry with them. When an agent migrates between sites, state information is stored inside a folder called `DATA`, and the code for the

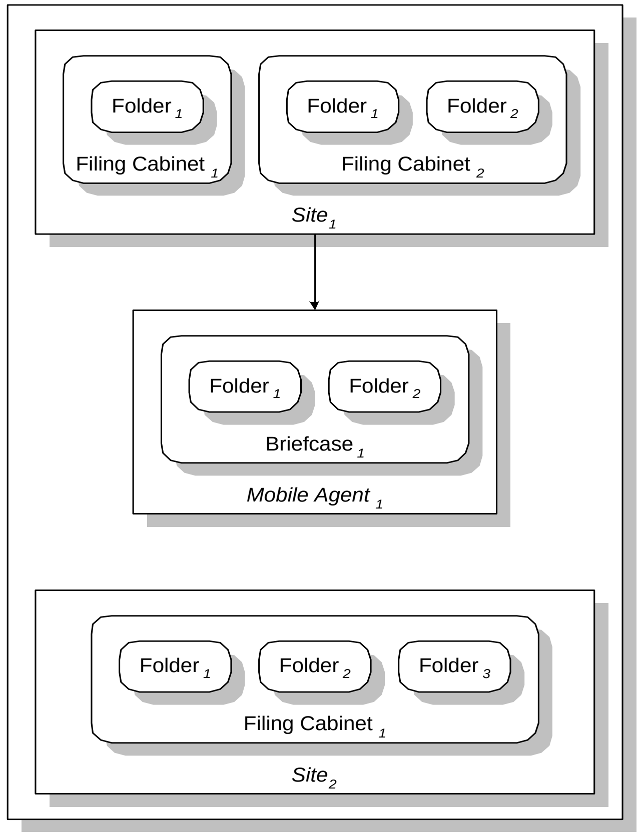


Figure 4.3: TACOMA Storage Objects

agent itself is stored within another folder called `CODE` (both within the same briefcase associated with that particular agent). At the receiving site, the code of the agent is extracted from the `CODE` folder and executed; the agent must look in the `DATA` folder for its data and state information.

There is only one abstraction for both communicating with other agents and transporting agents between sites; the `meet` operation. When an agent wishes to move to a new site, an entry is added to a folder called `HOST` in the agent's briefcase indicating where to move. This is a predetermined movement mechanism but the author's are attempting to remove the `HOST` folder by implementing a scheduler which partially automates the migration process.

To initiate the move, the agent `meet`s with the interpreter called `ag_tcl` on the remote site, thus:

```
meet ag_tcl briefcase
```

Similarly, when an agent wishes to communicate with another agent, it issues a `meet` request and passes a briefcase containing folders of relevant data to the receiving agent:

```
meet some_agent briefcase
```

The briefcase in this instance contains a special folder (`CONTACT`) which contains the name of the agent to be contacted. If the receiving agent wishes to respond, it issues a `meet` operation itself, passing a briefcase of data back. Thus, briefcases provide a level of abstraction for data since agents do not need to understand the entire contents of briefcases, just the folders in which they are interested. The entire communication mechanism is node-oriented and synchronous.

TACOMA also has mechanisms for firewall agents (`tag_firewall`) and RPC-style communicating agents (`tag_rpc`). The `tag_firewall` agent provides an entry point to a site and a convenient point to perform such functions as authentication, access control and accounting. `tag_rpc` allows client agents to block while they are awaiting results from a server agent.

The future of the TACOMA project lies in investigating mechanisms for fault tolerance, for example, by using rear guards. A rear guard is a special agent that is left behind when an agent migrates; it is responsible for launching a new agent should a failure cause an agent to vanish and for terminating itself when it is no longer necessary.

### 4.3.3 Telescript

Telescript (White, 1994) is a commercial product developed by General Magic Incorporated to support mobile agents for an electronic marketplace. Telescript the language (White, 1995) is an object-oriented programming language in which state-oriented migration is seen as the basic operation which is provided by the `go` instruction and a *ticket* argument that determines the destination site in "varying levels of specification". A Telescript *engine* exists at each site to accept and authenticate migrating agents and to restart the execution of agents at the statement immediately after the `go` command.

At the basic level, agents can move between *places* which are logical areas associated inside Telescript engines; an engine can support multiple places but has an outermost place called the *engine place*. Engine places themselves run within a larger context, known as a

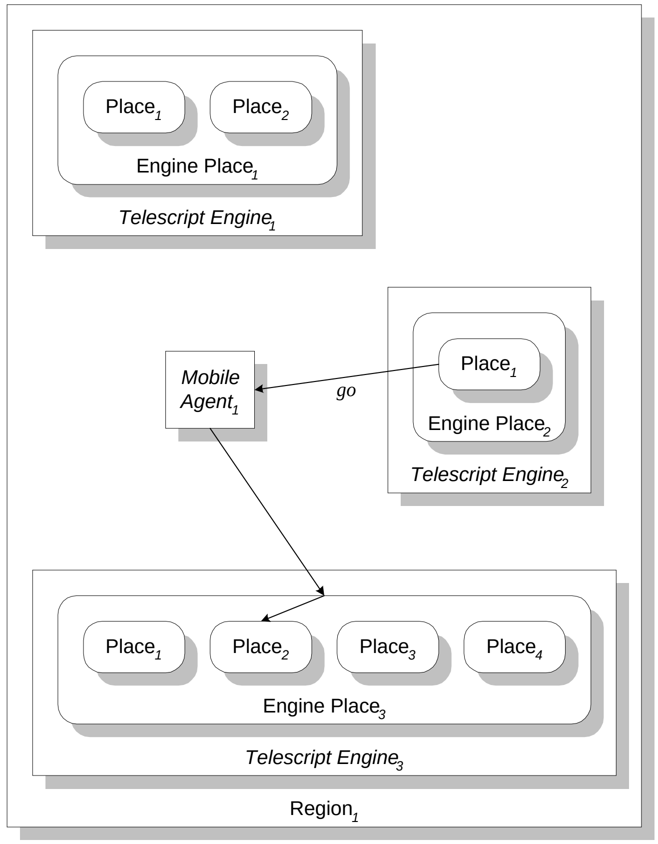


Figure 4.4: The Telescript Architecture

*region*, which are the connection points for the Telescript network. A region can consist of multiple engines running multiple places (figure 4.4).

Upon `go`ing to a place, the engine managing that place must authenticate the agent and either grant or deny access. If access is denied, then agents can be routed to a special place, called *purgatory*, where they are allowed to live for a short time in a limited environment. This allows them to recover and select a new destination.

However, if access is granted, then the agent enters that place and can communicate with other agents if the place is designated as a *meeting place*. The engine managing the meeting place mediates a meeting protocol between two agents in a synchronous and network-oriented manner, giving each agent the chance to accept or reject the meeting.

Telescript is currently being used in the AT&T PersonalLink™ network for mobile communications and is seen as a potential mobile computing mechanism for Personal Digital Assistants (PDAs).

### 4.3.4 An HTTP-Based Infrastructure for Mobile Agents

The HTTP-Based Infrastructure for Mobile Agents (Lingnau et al., 1995) is a project being developed at the Goethe-University in Germany. It is designed to provide a low-level infrastructure to support agent mobility and communication through the use of HTTP.

The architecture itself (figure 4.5) consists primarily of an *agent server*, which is a process that executes on every host that can be accessed by agents. Its tasks include accepting agents, creating an appropriate runtime environment for agents to execute within, supervising the execution of agents and terminating agents if required. In addition to this, the agent server must also organise the transfer of mobile agents to other hosts, manage communication between agents and their users, and perform authentication and access validation.

Mobile agents that move between agent servers are launched from a *home server*, which keeps track of the agent's progress through the network. Agents are transferred as encapsulated Multipurpose Internet Mail Extension (MIME) documents in a stateless manner, by `POST`ing them to a special URL that is managed by an agent server. Upon

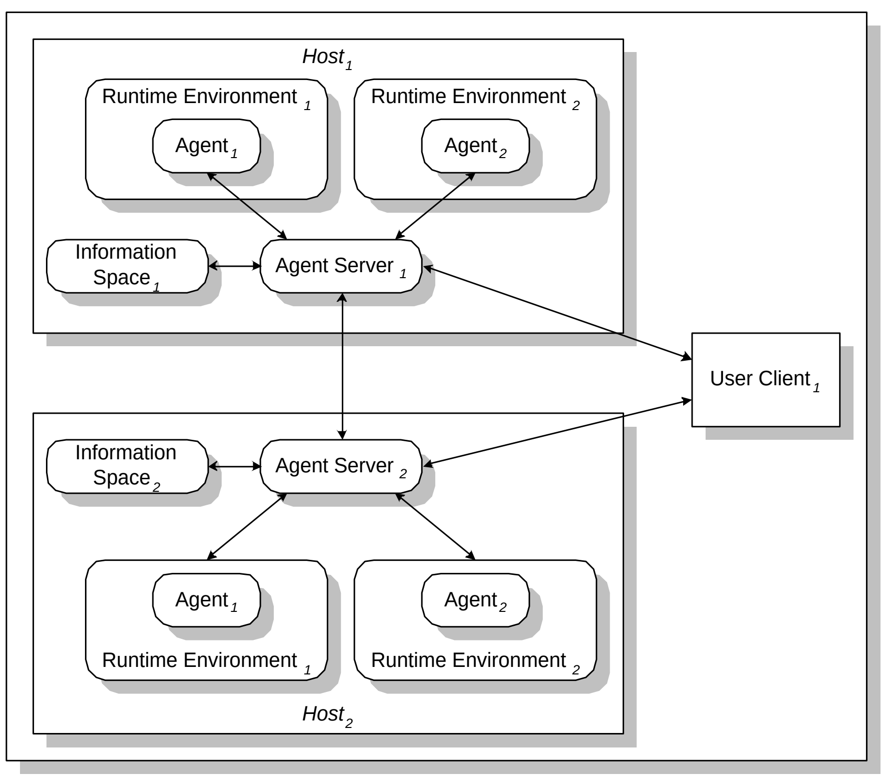


Figure 4.5: The HTTP-Based Mobile Agent Architecture

receiving this agent, the agent server parses the code of the mobile agent and determines whether it is acceptable. If the agent can execute on the site, then the agent server launches it within an appropriate runtime environment and forwards a URL to the home server of the agent to inform the user of its new location.

Communication between agents is also achieved through HTTP in an asynchronous manner only; each agent server maintains an information space which is accessible to agents via `POST`ing and `GET`ting. The agent server mediates access to entries within its information space by checking the requesting agent's credentials against the `Access:` header that was supplied with the entry.

At present, the framework is incomplete. For example, there is no provision for synchronous communication between agents, there is a painful lack of adequate security measures and agent servers must be aware of their neighbours to enable mobile agents to decide where to move to next.

### 4.3.5 Conclusions

Although each of these mobile agent architectures support differing levels of functionality, they each attempt to address the same problem, namely, enabling portions of code to execute on different machines within a wide-area network. Table 4.1 provides a comparison

| | | **Characteristics** | | | |
|---|---|---|---|---|---|
| | | **Migration** | **Acquisition** | **Route Determination** | **Communication** |
| **Mobile Agent Systems** | **Agent TCL** | State-oriented | None explicitly | Predetermination | Network-oriented, synchronous |
| | **TACOMA** | Stateless | None explicitly | Predetermination | Network-oriented, synchronous |
| | **Telescript** | State-oriented | Web integration | Dynamic determination | Network-oriented, synchronous |
| | **HTTP-Agents** | Stateless | Web integration | Hybrid determination | Node oriented, asynchronous |

Table 4.1: Comparison of Mobile Agent Architectures

of the features of each of the architectures compared against the characteristics outlined in section 4.2.

The weakest system is the HTTP-based infrastructure for mobile agents due to its primitive agent migration mechanism and simplistic communication model, for example, how can mobile agents pass real-time data between each other? However, asynchronous communication is desirable when an agent needs to issue multiple information requests and then process the results at a suitable time. The concept of a shared information space where agents can advertise their services is useful in arranging for mobile agents to meet and share data.

TACOMA also suffers from a stateless migration policy which hampers the ease at which migration can be achieved from the point of view of the programmer writing a mobile agent. Additionally, TACOMA's lack of dynamic determination means that a programmer must be aware of the locations that an agent can visit before it can be launched. This is advantageous for predicting the path and position of an agent, but limits its functionality. However, TACOMA's data storage model is very flexible and the actual way in which information is transferred between agents and servers is beautifully simple.

The main problem with the Telescript architecture is that it only supports the Telescript language (which *must* be entirely supported by Telescript engines) and is not open to researchers to develop. On the other hand, the Telescript language was designed for writing mobile agents so migration is well supported, security is intrinsic to the system and communication between Telescript agents is handled through meeting places and also directly between agents.

Agent TCL appears to be the most flexible architecture, since it supports state-oriented migration, multiple languages and networking protocols and a comprehensive communication subsystem. Once the predetermined routing of agents is resolved and some security measures are implemented, it will be a flexible and powerful mobile agent system.

One feature that all of these mobile agent architectures have currently failed to address is in defining a domain of applicability; they all concentrate on the mobility of agents rather than the integration of agents with information resources. General Magic, for example, has been recently criticised by the mobile agent community for concentrating the Telescript language too closely on the migration aspect and not considering integration with the desktop and third-party applications. To remedy this situation, General Magic are looking to integrate Telescript mobile agents into the desktop in a heterogeneous fashion by using the abstract windows toolkit (AWT) in the Java programming language. Further, they have recently produced a set of integration tools, called Telescript Active Web Tools, which provide class hierarchies to enable Telescript agents to integrate with Web servers and resources.

## 4.4 Summary

Although current mobile agent systems do not provide the solution to distributed information management inherently in their architectures, they can provide a framework upon which such a solution could be developed. The concentration of these architectures on the aspects of migration and communication seem to be at odds with the development of the stationary agent resource discovery systems identified in chapter 3.

The next chapter describes the requirements for a mobile agent architecture that can support, but is not limited to, distributed resource management. It is hoped that the future development of this architecture will provide the integration of distributed information resources and third-party applications through the use of an extensible and flexible mobile agent subsystem.

# 5 A Mobile Agent Framework to Support Distributed Information Management

"I had to quit my job to have time to read my email" — Adam Curry

## 5.1 Requirements

The last chapter has demonstrated that current mobile agent systems, whilst providing the necessary tools to support the migration of agents between network nodes and the communication between agents, do not provide a direct solution for addressing the problem of information overload.

Therefore, a framework is needed that will address a number of issues, simultaneously, to ensure that the requirements of a distributed information management system are met. This chapter presents the potential requirements of such a system in the context of a mobile agent architecture and also details the initial framework and components required to support mobile agents that perform distributed information management tasks.

### 5.1.1 Arbitrary Migration

To ensure that programmers are given the maximum flexibility in designing agents, the migration of agents should be transparent and reduced to a single instruction (for example, the `go` command as used in Telescript). Additionally, this instruction should be able to be issued arbitrarily and execution should continue with the command after the jump instruction. This means that the state of an agent must be encapsulate by this migration command, which will package the agent into a format suitable for network transmission. This will allow programmers to concentrate on the tasks of writing distributed information management agents.

With stateless migration systems, the burden of ensuring that the data that needs transferring with the agent is in a well-defined area (for example, a folder within a briefcase in TACOMA) lies firmly with the programmer of the agent. Even though stateless migration requires less effort on behalf of the migration mechanism, it is considered an undesirable requirement due to the fact that it places some of the migration task with the programmer.

### 5.1.2 Persistence

A key attribute of a mobile agent system is the concept of persistence; the longer an agent is active, the more nodes it can visit and the more influence it can have. Indeed, if an agent has the ability to learn and augment its knowledge-base, then as time moves forward, the agent will gain 'experience'. As the agent gains more experience, the more valuable it becomes to the user, since it has useful knowledge about the nodes that it has visited.

For example, consider a resource discovery agent that visits nodes to find out about a particular subject. As the agent issues queries at each node, it can retain information about the range of facilities available, the number of query results found, the average response time, etc. When it is issued with a new request from its user, it can use this information to make a decision about which nodes might be worth visiting first.

The experience that individual agents gain might also be pooled into a collective experience-base that is stored at the node where a user launches their agents. In this way, new agents can be equipped with an existing knowledge-base with which to begin their search.

### 5.1.3 Fault Tolerance

As mobile agents move across networks and nodes to fulfil their goals, their 'experience' and knowledge base may grow as will the data that they collect. To prevent this information from being lost, an agent must be protected from a number of situations that may occur in its lifetime:

- *Network transmission fault*. If an error occurs while the agent is in transit between nodes, then it must either be retransmitted or restarted at the side from which it was being transmitted. If it cannot be retransmitted or restarted, then it must be forwarded to the node where it originated, so that the information it contains can still be of use to the user or so that it can be restarted. Whatever the contingency plan, the data that the agent contains must be preserved in some manner.

- *Node failure*. If the agent is currently executing on a node that fails, then it should be possible to trace the agent back and retrieve a previous copy. This can be achieved by storing a copy of the agent at various nodes during its lifetime. This storage will obviously depend upon the facilities available at each node; some nodes may not desire versions of agents to be stored, especially if they are large. Another solution is for the agent to forward a copy of itself at regular intervals to the node where it originated. Node failure itself can be categorised as serious and non-serious. A serious node failure involves an unexpected error in the system which means that the node fails immediately and without prior warning: the result is that the agent's data is lost. A non-serious node failure is where there is prior warning of the node failing, for example a shutdown, and the agent has chance to migrate to another node (or it is automatically transferred to a backup node).

- *Security violation*. If an agent cannot be transmitted to a new location due to a security constraint (for example, the agent is not from a trusted node), then the agent should be restarted at the local node and given a chance to choose a new destination. This problem can be alleviated (but not totally solved) by requesting authentication in advance. For example, the agent might be rejected when it has been received and processed by some validation mechanism and the agent then attempts to execute a prohibited action. The action taken is the responsibility of the local node, since the agent might be a virus or a legitimate agent that has exceeded the bounds of its objectives. Potential solutions to this problem depend upon the severity of the security violation. If the agent attempts a prohibited action, it may be suspended and forced to execute in a very controlled environment so that it can choose another destination. However, if the agent fails a validation test, then is might either be destroyed (information and all), or destroyed and its information transmitted back to the node from where it originated.

### 5.1.4 Agent Communication Language

Agents should be able to communicate with other agents in a common language, which conveys meaning and represents information in a standard format. An example of such standards that are both part of the ARPA Knowledge Sharing Effort (Patil et al., 1992) are the Knowledge Query and Manipulation Language (KQML) and the Knowledge Interchange Format (KIF).

KQML (Finin et al., 1995) is a high-level protocol and language for agent to service and agent to agent interaction and communication, commonly termed Agent Communication Languages (ACL). It is based around a notion derived from speech act theory in linguistics (Austin, 1962) where messages that are communicated consist of *performatives* indicating what the receiver should do with the message, for example, `tell`. Currently, KQML provides performatives which deal with belief revision, querying, knowledge-base maintenance, actions and services. Also, KQML has the ability to nest performatives, such that more specialised performatives can be built from more general-purpose performatives.

KIF (Genesereth et al., 1992) is a rich language that provides a standard representation of information. It can express beliefs, rules, facts and partial descriptions of functions amongst other things. However, KIF is used for more than just a information representation; it also provides a central language for communication between other ACLs. Translating between $n$ ACLs and a central information language such as KIF, reduces the number of translators required from $O(n^2)$ to $O(n+1)$.

The advantages of using ACLs and standard information representations are clear; if all agents within the system conform to an interchange standard, then the amount of data translation is reduced and the introduction of translation errors into the data are lessened.

### 5.1.5 Communication

Another aspect that needs to be considered is how agents are to name and reference other agents? It is obviously not suitable to reference agents through network or machine address, due to the fact that they may move frequently. Additionally, some form of registration needs to take place when the agent moves, so that its current address can be resolved and located. Finin (Finin et al., 1995b) advocates the use of a hierarchical naming convention, not dissimilar to the Domain Naming Service (DNS). Each *agent domain* is responsible for maintaining a portion of the *agent naming tree* and addressing is resolved by *agent name servers*. More importantly, they illustrate the use of agent proxies to provide protocol gateways and firewalls.

The Heterogeneous Communication Model (Goose, 1995; Goose, 1996) provides a similar mechanism for agents to communicate with other agents through an addressing mechanism; as agents transport to a domain, they register their presence with a central *router*. The form of this registration determines which types of messages the agents is willing to receive: this is analogous to offering server-style functions to other clients. In an example scenario, an agent could register once to indicate that it had information on a particular subject and again to say that it required information on another subject. Due to the configurable and modular nature of the HCM, these concepts can be extended over domains by adopting a DNS-like architecture.

### 5.1.6 Heterogeneity

To ensure that a mobile agent system is generic enough to suit a wide range of applications, it must possess heterogeneity in the following areas:

- *Platform*. Architecture and operating system heterogeneity can be achieved by transmitting agents as source code and interpreting them at their destinations, or by pre-compiling them into byte code and interpreting the byte code at their destinations. The latter is more privacy-oriented, since it might not be prudent to allow the source code of an agent to be inspected. Also, byte code is less bulky than source code, can be interpreted faster and can be authenticated easier than either machine code or source code.

- *Network*. To ensure that the agent has access to as wide a range of resources as possible, it should support interchangeable network protocols. This can be achieved at a network level through the use of gateways, which equip agents with a suitable protocol module that allows them to exist on the other side of the gateway.

- *Language*. Agents should be able to be written in any language that is suitable to their task or with which the programmer is most familiar. However, to ensure that the agent offers the maximum flexibility, it must be transmitted in a form that can be interpreted rather than compiled. This has the implication that an interpreter has to be written or modified for each target language, to support the extra functionality of the mobile agent system. If this is not an option for some languages, then it may be that some will not be able to support all the features that are available, for example, an agent that has been written and compiled in C would not be heterogeneous enough to migrate to different platforms.

The advantages of heterogeneity for distributed information management agents are clear (White, 1994b); it broadens the scope of information resources that agents can access and control on behalf of their user and allows the agents to integrate with more distributed resources and applications.

### 5.1.7 Security

To ensure that agents can do no harm to their environment or to another agent, a number of security considerations need to be taken into account:

- *Permission*. The actions that an agent is able to perform should be regulated against a permission set for the intended recipient to ensure that they are allowed within the current context. This means that the permission set used to protect objects must be rich and flexible enough to ensure that its integrity is maintained from unauthorised accesses. Additionally, different nodes may assign different permission sets to similar objects, for example, on a node where security is very tight, all query accesses must be checked.

- *Authentication*. Before a permission system can be implemented, the identity of the agent must be confirmed and its point or origin established. If neither of these points can be determined, then the agent may be granted the lowest access permission or removed from the node, as detailed previously. However, authentication may be required once per access (for high security systems) or once per session (for lower security systems).

- *Transmission*. While the agent is in transit, it is vulnerable either to being modified in order to change its original purpose or to being duplicated to obtain its authentication signature. Therefore, when the agent is transmitted, it needs to be protected in some way. Encryption mechanisms such as PGP (Garfinkel, 1994) can be used to protect the agent at a data level, and mechanisms such as the Secure Sockets Layer (SSL) (Freier et al., 1995) at a network level.

- *Verification*. When the agent arrives at a node, certain verification tests can be made upon its byte code to ensure that it will neither perform malicious acts nor prohibited acts. This type of verification is performed rigorously by Telescript engines and less complete security is provided by the validation component of a Java-aware browser.

- *Charging*. As an agent interrogates information, it may be charged for access on the amount of data it requests, or once per access. Either way, some system needs to be employed to ensure that the agent has funds to pay for services, that those funds belong to the agent and the agent belongs to the right user. Existing technologies, such as the Electronic Cash Unit (ECU) (Chaum, 1992) can ensure that mobile agents have to pay for the services that they use.

The discussion surrounding the security of mobile agents is currently a topic which generates much debate (Chess et al., 1995). There is genuine concern that mobile agents could present a fundamental problem to networks and systems unless their movement and execution are closely monitored and regulated. It is obvious that some forms of control will have to be implemented to ensure that mobile agents cannot execute on prohibited nodes, cannot monopolise resources or cannot perform malicious acts across network nodes.

One policy involves giving an agent exactly the right amount of permission required to complete its pre-authorised task. If an agent requires more permission, it must apply to the regulatory body of the node. Access or denial will depend upon a number of factors; the ability of the agent to pay for the new permission and the trustworthiness of the agent in the past, as examples.

### 5.1.8 Summary

The requirements that have been previously outlined are considered the minimum necessary to build an initial framework for mobile agents to support distributed information management tasks.

It is hoped that requirements such as arbitrary migration and ACLs will give the programmer flexibility when writing agents, heterogeneity and communication will allow an agent to reach as many distributed information resources as possible, and security and fault tolerance will make the mobile agent system both trustworthy and robust.

## 5.2 The Framework

The framework that is to be presented is based around an agent-oriented model; everything in the system is abstracted through agents (figure 5.1). The system is comprised of both static and mobile agents; *static agents* provide resources and facilities to mobile agents and *mobile agents* move between network domains taking advantage of these resources to fulfil

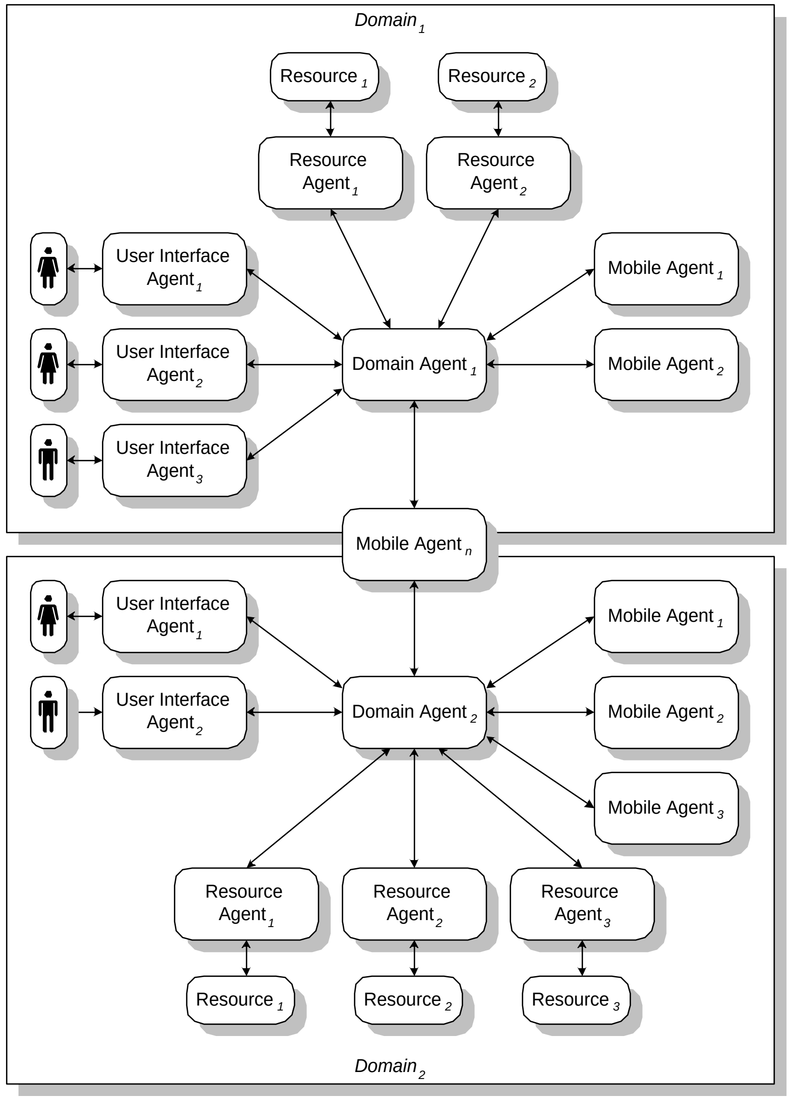


Figure 5.1: Mobile Agents for Distributed Information Management Framework

their goals.

It is envisaged that mobile agents will migrate in a state-oriented fashion and be comprised of code, state and some form of knowledge base. The code segment of an agent will describe the functionality of the agent, for example a resource discovery agent or a link maintenance agent. The state segment will include the stack and local data that the agent is currently executing upon; this will be transferred with the agent transparently. Finally, the knowledge-base will contain the 'intelligence' of the agent, that is, its experiences, its goals and its beliefs. This segment will also contain temporary information that the agent is working upon before it is returned to the user.

The manner in which mobile agents (and indeed, static agents) achieve the requirements identified earlier forms the discussion in chapter 6, Future Work. The next sections take each component of the framework and detail its general activities and purpose in the mobile agent infrastructure.

### 5.2.1 Domain Agent

A *domain agent* is a static agent that supervises the activities that occur within a domain; a *domain* is a logical boundary used to delimit nodes, agents and resources into manageable and distinct entities. The domain agent has a number of responsibilities to information resources, users and agents within a given domain:

- Providing a migration service to mobile agents wishing to leave the domain; the domain agent is responsible for ensuring that the mobile agent is transferred successfully. Upon valid exit, notification should be sent to the node which originally launched the agent, to enable a user to keep track of their mobile agents.

- Providing an authentication and validation check on mobile agents wishing to enter the domain; agents that cannot be authenticated or fail the validation check are rejected. Upon valid entry, notification should be sent to the node which originally launched the agent, to enable a user to keep track of their mobile agents.

- Launching received mobile agents in a suitable runtime environment, which will be related to the amount of trust given to an agent; agents from trustworthy nodes may be allowed access to more information resources than agents from unknown nodes. A suitable runtime environment will depend upon the language that the mobile agent is written in, the access level it is allocated and the functions it wishes to perform.

- Mediating global access to information resources; before a mobile agent can access an information resource it must obtain permission from the domain agent. Depending upon the policies imposed on the domain agent, public resources might grant read access automatically. However, for read access to restricted information or modification access, the domain agent must approve and grant access.

- Providing a central message router; all message interactions between static and mobile agents are routed through the domain agent. This allows agents to communicate in an asynchronous fashion. However, once two agents have become aware of each other's presence (through communicating initially through the domain agent), they can initiate a direct and synchronous communication between themselves, to transfer high-bandwidth data for example.

- Providing a central registration area where static and mobile agents can register and advertise their resources and interests.
- Ensuring that the domain does not become swamped by mobile agents. This can be achieved by limiting the amount of agents that can exist within the system at a given time and implementing a charge for processor cycles and resource access. This way, even if a mobile agent attempted to hog a resource, it could only do so for as long as its credit limited lasted.
- Advertising public information resources and the presence of mobile agents to enquiring agents. In this way, mobile agents can interrogate the contents of a particular domain before moving, to ensure that it contains information resources will help it to achieve its goals. It also provides a rudimentary mechanism for agents to locate each other.
- Periodically announce their presence (through broadcasting or multicasting) to the larger community of domain agents. This will allow each domain agent to maintain a list of other domains that is available for mobile agents to use in determining their next jump point.

The domain agent, then, is the key force within a domain and is ultimately responsible for ensuring that security is enforced within the domain and that agents can communicate with information resources and other agents.

### 5.2.2 Resource Agent

*Resource agents* are static agents that exist within a domain to provide a level of abstraction between mobile agents and the information resource to which they provide access. The purpose of a resource agent is to mediate access to a particular resource at a local level for an agent; the resource agent understands how to interrogate the resource and also understands the permission structures associated with the resource.

The resource agent has a number of functions to fulfil:

- To be fully conversant in the protocols of the information resource to provide complete access; the information resource should be completely accessible by agents, so that user intervention directly on the documents is not required. This would imply that the user has a set of distributed information management mobile agents that are able to manage and control their information resources through resource agents.

- To provide a standard and generic set of services which all mobile agents can utilise, for example, `query`, `get`, `delete`, `add`, `alter`. The more specialist functions of an information resource, for example linking, would be handled by other service primitives that would imply that the mobile agent understood some of the functionality and capabilities of the information resource. It would be desirable to decouple this relationship entirely, that is, to provide an extensible set of services across information resources. This could be achieved by attributing the functionality of a service to an associated specification of its behaviour, rather than its descriptive title.

- To mediate access to the resource on a per agent basis; once an agent has been granted access to an information resource, it is the task of the resource agent to ensure that the integrity and privacy of the resource is maintained. To this end, once the agent enters the domain, the domain agent should allocate the mobile agent a permission set which is recognised locally by the individual resource agents.

- To provide conversion between the information representation of the mobile agent and the information representation of the information resource that the resource agent is managing.

- To advertise the presence of the information resource by registering the services available to the domain agent. In this way, mobile agents can be aware of what information resources are available within a domain (by querying the domain agent) before they arrive at a domain.

The interaction between resource agents and mobile agents will form the crux of the distributed information management aspect of the infrastructure. The flexibility that mobile agents are afforded in accessing these information resources and the manner in which they interpret the results will determine their usefulness to the user.

### 5.2.3 Mobile Agent

*Mobile agents*, as their name suggests, are the components within the framework which can migrate between network nodes. They are the mechanism by which the user exercises control over their own distributed information resources and gain access to other, shared information resources.

Mobile agents are equipped with a set of user-defined goals which describe the nature and limits of their functionality, for example, a resource discovery agent is a mobile agent with a different goal set than, say, a navigation assistant agent. It is probable that, in addition to the limits on the functionality that users place on their mobile agents, the mobile agents themselves will encounter limits (in the form of security, authentication and validation) that exist within domains. In some cases, these limits (for example, lack of funds to pay for resources) will compromise the goals that have been given to the mobile agent.

There are two general approaches to the coordination of mobile agents:

- *Single-agent systems*. Within single-agent systems, the functionality of the system is expressed in terms of solitary agents; mobile agents are allocated tasks and must achieve those tasks personally, albeit through interaction with other agents. Whilst this is limiting in the tasks that an agent can fulfil and can incur latency in the turnaround of results, it does make that task of programming the framework easier since only agent to agent communication has to be considered. To alleviate the latency problem, agent cloning can be instantiated to allow agents to create replications of themselves. This would allow agents to cover a much wider area of information space in a shorter time than a solitary agent crossing the same amount of space.

- *Multi-agents systems*. These systems differ from single-agent systems in the fact that a mobile agent can spawn other related agents. Hierarchies of communicating and cooperating static and mobile agents can be constructed to assist the original agent in fulfilling its goals. This field is closely related to the principles of Distributed AI (as mentioned in chapter 3) and is more suited to complex problem solving than distributed information management.

The essential functions of mobile agents will be delineated by the distributed information management tasks that they are allocated. However, they have the following interactions with the framework:

- Mobile agents determine where to migrate to next by initially querying the local domain agent for a list of domains of which it is aware. The mobile agent can then use this information to contact each domain agent and determine which one offers a compatible set of information resources.

- Mobile agents are authenticated by an electronic signature that they carry; this signature is encrypted and is verifiable with the host domain of the mobile agent (which implies that the host domain is also authenticated in some manner). Additionally, mechanisms must be employed to ensure that mobile agents are not compromised during transit or within a domain and that the behaviour of a given mobile agent is consistent with the behaviour that it was given when it was launched. Due to the fact that it is impossible to guarantee these requirements, since a malicious agent may be launched from an authentic domain, mobile agents should be taken on face value and given the minimum amount of permission possible.

- Mobile agents possess the characteristic of persistence; they use migration as a mechanism to achieve longevity. Mobile agents must learn from the domains that they visit, the actions that they perform and the results they receive by augmenting their knowledge-base accordingly. Therefore, the 'older' a mobile agent is, the more 'experience' it will have gained; this experience can become invaluable to new 'raw' agents.

- Mobile agents may communicate with other mobile agents to achieve their goals. There are a number of advantages to this approach; in the global information space, certain mobile agents will be performing the same or similar tasks and therefore it makes sense if they can trade information. Additionally, mobile agent might trade other services, such as experiences of productive domains for electronic cash or information. However, the disadvantages of mobile agent communication rationalises to one of trust; is it possible to trust the information given by one mobile agent to another, especially when electronic cash is concerned?

- Mobile agents should transmit the results of their findings and actions regularly to their user on the host domain. This way, the user can ensure that the mobile agent is functioning correctly and also helps prevent the mobile agent from becoming too big (in terms of size, memory and processing requirements) to move or to process.

It is clear then, that mobile agents are the ultimate effector of change within a distributed information management environment, and as such, should be treated cautiously. The possibility for mobile agents to effect changes or gain access to private information must be closely monitored by the domain agent and the individual resource agents. A comprehensive authentication, validation and access permission system is vital to ensure that malicious agents can do no harm to a system and that legitimate mobile agents can fulfil their goals.

### 5.2.4 User Interface Agent

The *user interface agent* is an agent that resides within a domain and provides a level of abstraction for the user away from the details of the mobile agent framework. It is essentially an interface agent (see chapter 3) that needs to be capable of the following tasks:

- To launch distributed information management mobile agents on behalf of the user and to track their progress and position.

- To provide mobile agents with a communication point through which they can return the results of their tasks. User interface agents may need to possess local mobility to allow for node failures, since they need to be executing even when a user is disconnected from the system.

- To organise and preprocess information returned from mobile agents into a form that is suitable to the user. This may involve filtering out replicated information, presenting urgent information to the user quickly, rendering information using the tools that the user is most familiar with, etc.

- To provide domain agents with the authentication credentials of the mobile agents of a user. It is the task of the domain agent to ensure that the requesting entity is itself an authentic domain agent, to make certain that authentication credentials are not given to unauthorised systems.

User interface agents provide the user with a window onto their agents, their status, their results and the mobile agent framework. The creation of new distributed resource management agents could either be the task of the user to write (or reuse with different parameters), or the task of the user interface agent to interpret the requirements of the user and generate mobile agents to fulfil those tasks automatically. The latter approach is most desirable, but is still a long way off given the technology that exists currently.

### 5.2.5 Gateway Agent

A *gateway agent* is a static entity that resides within a domain and provides access to another network or to a restricted domain. Therefore, gateway agents need to provide the following services:

- *Network heterogeneity*. This can be achieved by equipping mobile agents with interchangeable protocol modules; these modules would allow an agent to move between networking protocols. It would be the task of a gateway agent to grant the mobile agent access to the new network and to remove old networking modules and add new ones to enable the mobile agent to communicate in the new network.

- *Firewalling*. A domain, or set of domains, can still exist within the same networking protocol, but be protected from unauthorised access at a higher level than the domain. The gateway agent would filter all incoming and outgoing agents to ensure that they did not contain malicious intent or were not trying to leave with sensitive material.

## 5.3 Summary

The framework that has been presented in this chapter has shown that there is much work that needs to be undertaken above and beyond what current mobile agent systems offer for distributed information management. The framework provides a network-wide environment in which users can manage, augment and integrate their existing distributed information systems, and also discover and incorporate additional bodies of information from other distributed information systems that have been made accessible by other users. The sharing permissions that these information resources are issued with will determine the level at which anonymous mobile agents can interact with foreign resources; the potential for cooperative working is obvious.

The realisation of this framework will take two stages; production of the infrastructure components and the development of a set of mobile agent-based tools to support the user in achieving distributed information management goals. The next chapter identifies some of these issues as future work and discusses their potential implementations.

# 6 Future Work

"Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and perhaps weigh 1½ tons" — Popular Mechanics, March 1949

## 6.1 Framework Implementation

The first step in developing a prototype is to build the framework for mobile agents to execute within. It is envisaged that this prototype framework will be implemented in two phases. The first phase will implement the core infrastructure of agents (mobility, communication, integration) and the second stage will deal with the aspects of distributed information management of both mobile and static agents.

### 6.1.1 Agent Framework Architecture

The framework described in chapter 5 was not designed around any one mobile agent implementation. This decision was made so that, theoretically, any of the mobile agent architectures described could be adopted and modified to support mobile agents for distributed information management.

For this reason, a choice needs to be made; whether to advocate the use of an existing mobile agent system (such as Agent TCL), or whether to develop a new mobility infrastructure from scratch. The arguments for reusing an existing system are persuasive, since the underlying migration and communication layers are already developed and available. However, the system that is chosen should exhibit the following characteristics:

- *State-oriented migration*. The reduction of migration to a single instruction that packages the agent automatically, transfers the agent and restarts the agent on the instruction after the movement imperative.

- *Extensible architecture*. To allow new features to be added, for example, extended security and other communication models. Additionally, the system chosen should allow new protocols and languages to be integrated seamlessly; unlike Telescript which only supports the Telescript language.

- *Flexible security model*. To allow the security model to change according to the environment. Since this is primarily a research exercise, a heavy security model imposed from the outset, as employed by Telescript for example, would hinder the development of the framework.

- *Inherent heterogeneity*. To allow the mobile agent framework to be accessible on as many platform types and across as many networks as possible, but some platforms may not possess the ability to support all features of the framework. Indeed, for platforms which cannot maintain a rigid security model, provision needs to be made within the framework to accommodate them, albeit at a very low security level.

The development of the framework from scratch also has a number of advantages: more control over the system and its implementation and the ability to tailor the framework closely to the problem being solved. However, it would a considerable task to develop a migration policy and determine a heterogeneous data format that would allow mobile agents to be migrated transparently.

Instead, it would make more sense to take advantage of existing technology in this area, namely Distributed Object Management (DOM). With such systems, an object (in this case an agent, its data and its knowledge-base) could be distributed automatically by a DOM sub-layer. Examples of such systems are DMeroon (Queinnec, 1995) and HORB for Java (Satoshi, 1996). They present an attractive alternative and would considerably ease the development of the framework from scratch.

### 6.1.2 Agent Programming Language

Once an architecture has been selected, then a programming language will need to be chosen to implement the framework. It is commonly accepted that interpreted languages, often referred to as scripting languages, represent the best implementation to achieving heterogeneity and enforcing validation principles. This helps to make the agents of the infrastructure portable and dynamic, and gives the system greater control over their execution. Languages that are compiled to the machine code of a specific system (for example, C) are difficult to make dynamic and portable to other platforms and languages that are interpreted direct from source code present a potential security risk.

The following is a brief summary of the current interpreted languages that hold the most promise for developing the agent framework:

- *Java* (Gosling et al., 1995). Java, developed by Sun Microsystems Incorporated, is an object-oriented language that is very reminiscent of C++ but removes some of the more contentious issues, for example, pointer arithmetic. Java code is compiled to platform-independent byte code for portability, but migration and dynamic extensibility of byte code are not explicitly supported. The object-oriented nature of the language makes it highly desirable, since a generic agent class could be developed and other agent types (for example, a domain agent) could be specialisations upon that class, for example, the JavaAgentTemplate (Frost, 1996).

- *Telescript* (White, 1995). The Telescript architecture, as developed by General Magic Incorporated, is also an object-oriented programming language. However, Telescript has inbuilt support for code migration and explicit support for the dynamic extensibility of objects. A compiler translates Telescript source code into an interpreted language that can be processed by Telescript engines, although the authors stress that it this not a scripting language in the traditional sense. The Telescript language is very useful, since generic agent types have already been created within the class hierarchy, so the development of new agents is relatively straight-forward.

- *Tool Command Language* (Ousterhout, 1994). TCL (pronounced 'tickle') was originally designed to perform the tasks of traditional scripting languages; the creation of macros or code segments that link compiled applications together. However, more recently, TCL has been proposed as a language for writing mobile agents (Gray, 1995; Gray, 1995b). Unfortunately, since TCL is a scripting language, its inherent support for migration and dynamic extensibility is non-existent. Also, since the language is interpreted directly from source code it is also disadvantaged, due to the fact that it may not be wise to allow other people to inspect the source code of agents. However, despite these disadvantages, TCL is being used and a proposal has been put forward for a safe version of the language, called Safe TCL (Ousterhout, 1995).

- *Obliq* (Cardelli, 1995). Obliq is a lexically-scoped, interpreted language that supports distributed object-oriented computation. The language achieves distribution of network objects through distributed lexical scoping; the concept of binding network addresses and identifiers to their original network hosts, rather than the host upon which the object is currently executing. Obliq has intrinsic support for what it calls 'agents': distributed computations that can be migrated across networks. However, as with TCL, agents are migrated as source code that is interpreted at remote engines.

It would appear then, that object-oriented languages which compile to a platform-independent byte code or an interpreted language hold the most promise for the future development of the framework implementation. However, although a very desirable language, Telescript is hampered by the fact that it cannot be used outside of the Telescript environment and it has no support for heterogeneous integration with user interface environments: an area where Java excels.

### 6.1.3 Agent Communication Language

It is important that all of the agents within the framework communicate in a single information representation and use a common communication mechanism (called an Agent Communication Language (ACL)). As has already been described in chapter 5, KQML (Finin et al., 1995) offers a standard communication language between agents and KIF (Genesereth et al., 1992) offers a standard information representation that agents can pass between themselves. This reduces the translators required to two types; resource agent to local information resource protocol (for example, KIF to WWW) and resource agent to user-readable format (for example, KIF to text).

Other ACLs that need to be investigated are:

- *AgenTalk* (Kuwabara, 1995). AgenTalk is a coordination protocol description language for multi-agent systems which allows coordination protocols to be defined incrementally and to be easily customised to suit application domains by incorporating an inheritance mechanism.

- *Agent-Oriented Programming* (Davies et al., 1994). AOP is a specialisation of object-oriented programming that supports three mental states; *belief*, *capability* and *commitment* which are represented by *commitment rules*. Message types (`inform`, `request` and `unrequest`) provide the mechanisms for agents to communicate. Once a rule is triggered within an agent (that is, when a match is made between a message type and the internal beliefs of an agent), the agent will generate a commitment to either *do* an action, *refrain* from doing an action or *inform* an agent of something.

Currently, the most promising ACL appears to be KQML, as it is developing as a standard and is also being adopted by a variety of agent development projects.

## 6.2 Distributed Information Management Agents

Once the initial framework is in place, then agents for distributed information management can be developed. These agents need to express some of the characteristics described in chapter 3 to ensure that they can fulfil their goals sensibly (*intelligence*), react to their environment (*reactiveness*), adapt to their environment (*learning*), take the initiative (*autonomy* and *pro-activeness*) and move closer to the distributed information resource (*mobility*). This will help to ensure that a trust relationship is developed with the user.

The knowledge-base of an agent will be written in a reusable knowledge format, such as KIF, and agents within the system will communicate through KQML performatives; this gives the added advantage that the framework will also be able to communicate with other KQML-based systems. It will be the task of the user interface agent and the resource agent to perform translations upon KIF information to convert it to and from the local protocol format; it will be the task of mobile agents to perform processing on information received from resource agents.

Agents, such as user interface agents, have the potential to be developed in a number of ways. Essentially, they could be implemented as an interface agent (chapter 3) which sits along side the user and 'learns' from their actions; in this manner, the user interface agent can take the initiative and autonomously launch mobile agents of its own to perform functions that it thinks (hopes) will be of use to the user, dependent upon the actions that the user is performing or has performed or dependent upon changes in the user's environment (for example, the arrival of an email).

The user interface agent also has the potential to integrate with the user's desktop; this would give it the ability to coordinate its activities with other, non-distributed information resources that are available on the local network, for example, email, network news, etc.

Moreover, the user interface agent could be written to integrate with a series of desktop applications (through interchangeable modules), for example Eudora, Netscape, Elm, etc.

Mobile agents for resource discovery can develop on the ideas that exist for current discovery mechanisms; mobile agents for navigation assistance can develop upon the ideas expressed by Wilkins (Wilkins, 1994); mobile agents for system integration can take the integration ideas expressed by Davis et al. (Davis et al., 1994) for integrating the Microcosm open hypermedia system into the desktop. However, mobile agents for information integrity will require more research, since information management in distributed information systems and on a global scale is limited at present.

# 7 Conclusions

"Include me out" — Sam Goldwyn

The growth of inter-networking that has recently taken place has demonstrated that the electronic information systems of today are providing the user with more electronic information than has ever been previously available. As more organisations and users 'get on-line', then not only will their needs and expectations increase, but the burden on current and new systems will increase, expecting them to provide more information resources.

Internet users of today are given relatively few tools to assist them in navigating through the plethora of distributed information systems that exist. The most rapidly expanding protocol currently within the Internet is the World Wide Web (Web) and the tools that people have developed for it are firmly centered around the resource discovery area; there are almost no management tools that exist for controlling and manipulating distributed Web resources.

Distributed information management is a term that is being used more frequently to describe a set of tools that provide four basic functions; resource discovery, information integrity, navigation assistance and system integration. These functions assist the user in navigating and managing distributed information resources, such as the Web, FTP, Gopher and Hyper-G. It is proposed that if these tools were available, the user would be better served by distributed resources.

However, most current tools are purely manipulation tools which require the user to maintain some form of interactive control over them, for example, an FTP session. Agent technology is a step away from direct and interactive tools into the realm of task delegation; agents are essentially processes that possess characteristics that enable them to perform tasks that would normally be carried out by the user.

These characteristics define the nature and limitations of agents and also enables the user to predict the behaviours of their agents. One of the most important features that agents must exhibit is trustworthiness; an agent that performs its task badly is of little use to a user if they not only have to perform the task by hand, but also have to repair the damage caused by the agent in the first place.

To assist agents in completing the tasks that users assign to them, it is proposed that they should possess the meta-characteristics of intelligence and learning. Intelligent agents can make reasoned and informed decisions based upon current knowledge; intelligent, learning agents can make reasoned and informed decisions based upon current knowledge and past experiences. These two facets are seen as crucial in agents gaining the trust of users.

Mobile agent technology is receiving much current interest and is seen as the next evolution in programming after the object-oriented paradigm. Mobile agents make a significant contribution to accessing distributed information in a natural fashion and for building applications and then integrating them into a distributed environment. However, contemporary mobile agent systems have put aside the more traditional view of agents (intelligence, learning, etc.) and have concentrated on the aspects of agent migration, agent to agent communication and the security, authentication and validation of agents. This technology, whilst promising for the future is not inherently realisable, since an important layer is missing between integrating mobile agent systems with distributed information systems, the desktop and (ultimately) the user.

The research presented in this thesis has advocated building a two-tiered approach to bringing mobile agent technology and distributed information management to the user; a lower strata dealing with process-level concepts (migration, communication, security) and an upper strata to handle the distributed information management tools, represented as component mobile agents. This approach allows mobile agents to reside beside the application *and* across distributed applications.

The framework that has been described attempts to provide a solution to the upper strata; it details how various agent types can integrate with distributed resources, users and other agent to provide the fundamental requirements of distributed information management. It is the future work of this research to develop a prototype lower strata upon which an upper strata can be developed and the integration of distributed information systems tested. It is further hoped that the development of mobile agent-based distributed information management tools will provide the user with the methods necessary to manage, navigate and discover distributed information resources that exist within the global information space that is developing.

# Bibliography


Agre, P. and Chapman, D., (1987) PENGI: An Implementation of a Theory of Activity. *In*: Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, USA, pages 268-272.

Andrews, K., Kappe, F. and Maurer, H., (1995) Serving Information to the Web with Hyper-G. *In*: Computer Networks and ISDN Systems, 27 (6), pages 919-926.

Anklesaria, F., McCahill, M. P., Lindner, P., Johnson, D., Torrey, D. and Alberti, R., (1993) The Internet Gopher Protocol, Request for Comments 1436, Ohio State University. *Available from*: `http://www.cis.ohio-state.edu/htbin/rfc/rfc1436.html`

Anklesaria, F., Lindner, P., McCahill, M. P., Torrey, D., Johnson, D. and Alberti, R., (1993b) Gopher+: Upward Compatible Enhancements to the Internet Gopher Protocol. *Available from*: `gopher://boombox.micro.umn.edu:70/00/gopher/gopher_protocol/Gopher+/Gopher+`

Apple Computer Incorporated, (1987) Macintosh Hypercard User's Guide.

Armstrong, R., Freitag, D., Joachims, T. and Mitchell, T., (1995) WebWatcher: A Learning Apprentice for the World Wide Web. *In*: Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments, Menlo Park, USA.

Austin, J. L., (1962) How to do Things with Words, Oxford University Press.

Balabanovic, M., Shoham, Y. and Yun, Y., (1995) An Adaptive Agent for Automated Web Browsing. *In*: Journal of Image Representation and Visual Communication, Volume 6.

Bates, J., (1994) The Role of Emotion in Believable Agents. *In*: Communications of the ACM, 37(7), pages 122-125.

Berners-Lee, T., Cailiau, R., Groff, J. and Pollermann, B., (1992) World Wide Web: The Information Universe. *In*: Electronic Networking: Research, Applications and Policy (Spring), 2(1), Westport, USA, Meckler Publishing, pages 52-58.

Berners-Lee, T. and Conolly, D., (1995) Hypertext Mark-up Language Specification 2.0. *Available from*: `http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec_toc.html`

Berners-Lee, T., Fielding, R. and Frystyk, H., (1995b) Hypertext Transport Protocol Specification 1.0 Internet draft 04. *Available from*: `http://www.w3.org/pub/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html`

Bloomer, J., (1992) Power Programming with RPC, O'Reilly and Associates.

Bond, A. H. and Gasser, L., *Eds*., (1988) Readings in Distributed Artificial Intelligence, San Francisco, USA, Morgan-Kaufmann Publishers.

Brooks, R. A., (1986) A Robust Layered Control System for a Mobile Robot. *In*: IEEE Journal of Robotics and Automation, 2(1), pages 14-23.

Brooks, R. A., (1990) Elephants Don't Play Chess. *In*: Designing Autonomous Agents, Maes, P., *Ed*., The MIT Press, pages 3-15.

Brooks, R. A., (1991) Intelligence without Reason. *In*: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, Sydney, Australia, pages 569-595.

Brooks, R. A., (1991b) Intelligence without Representation. *In*: Artificial Intelligence, 47, pages 139-159.

Bush, V., (1945) As We May Think, Atlantic Monthly, pages 101-108, July.

Cardelli, L., (1995) A Language With Distributed Scope. *In*: Principles of Programming Languages, January.

Carr, L. A., De Roure, D. C., Hall, W. and Hill, G. J., (1995) The Distributed Link Service: A Tool for Publishers, Authors and Readers. *In*: Proceedings of the Fourth International World Wide Web Conference: The Web Revolution, Boston, USA.

Castlefranchi, C., (1995) Guarantees for Autonomy in Cognitive Agent Architectures. *In*: Intelligent Agents: Theories, Architectures and Languages, Wooldridge, M. and Jennings, N. R., *Eds*., Lecture Notes in Artificial Intelligence, Volume 890, Springer-Verlag, pages 56-70.

Cate, V., (1992) Alex—A Global Filesystem. *In*: Proceedings of the USENIX Filesystems Workshop, pages 1-11.

Chaum, D., (1992) Achieving Electronic Privacy. *In*: Scientific American, 267(2), pages 96-101, August.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C. and Tsudik, G., (1995) Itinerant Agents for Mobile Computing, IBM Research Report RC 20010.

Davies, W. H. E. and Edwards, P., (1994) Agent-K: An Integration of APOL and KQML. *In*: Proceedings of the CIKM'94 Workshop on Intelligent Agents, Finin, T. and Labrou, Y., *Eds*., Maryland, USA, December.

Davis, H. C., Hall, W., Heath, I., Hill, G. J. and Wilkins, R. J., (1992) Towards an Integrated Information Environment with Open Hypermedia Systems. *In*: Proceedings of the Fourth ACM Conference on Hypertext, Lucarella, D., Nanard, J., Nanard, M. and Paolini, P., *Eds*., Milan, Italy, ACM Press, pages 181-190.

Davis, H. C., Knight, S. J. and Hall, W., (1994) Light Hypermedia Link Services: A Study of Third Party Application Integration. *In*: Proceedings of the European Conference on Hypertext ‘94, Edinburgh, Scotland, ACM Press, pages 41-50, September.

De Roure, D. C., Hall, W., Davis, H. C. and Dale, J., (1996) Agents for Distributed Multimedia Information Management. *In*: Proceedings of the First Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK.

Dennett, D. C., (1987) The Intentional Stance, MIT Press, Cambridge, USA.

Edwards, P., (1995) Intelligent Agents=Learning Agents. *Presented at*: BCS-SGES/DTI-ISIP Intelligent Agents Workshop, Oxford Brookes University, November.

Emtage, A. and Deutsch, P., (1992) Archie—An Electronic Directory Service for the Internet. *In*: Proceedings of the USENIX Winter Conference, pages 93-110.

Falcone, J. R., (1987) A Programmable Interface Language for Heterogeneous Systems. *In*: ACM Transactions on Computer Systems, 5(4), pages 330-551, November.

Ferguson, I. A., (1992) TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents, PhD Thesis, Clare Hall, University of Cambridge, UK.

Finin, T., Labrou, Y. and Mayfield, J., (1995) KQML as an Agent Communication Language. *In*: Software Agents, Bradshaw, J., *Ed*., MIT Press, Cambridge.

Finin, T., Thirunavukkarasu, C., Potluri, A., Mckay, D., and Mcentire, R., (1995b) On Agent Domains, Agent Names and Proxy Agents. *In*: Proceedings of the ACM CIKM Intelligent Information Agents Workshop, Baltimore, USA, December.

Flohr, U., (1995) Hyper-G Organises the Web, Byte, pages 59-64, November.

Freier, A. O., Karlton, P. L. and Kocher, P. C., (1995) The Secure Sockets Layer Specification 3.0. *Available from*: `http://home.netscape.com/newsref/ssl/0-INDEX.html`

Frost, R., (1996) Java Agent Template Version 2.0—Programmer's Documentation. *Available from*: `http://cdr.stanford.edu/ABE/documentation/progdoc.html`

Galliers, J. R., (1988) A Theoretical FrameWork for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict, PhD Thesis, Open University, UK.

Garfinkel, S., (1994) PGP: Pretty Good Privacy, O’Reilly and Associates.

Genesereth, M. R. and Fikes, R. E., (1992) Knowledge Interchange Format Version 3.0 Reference Manual, Technical Report Logic 92-1, Stanford University, Stanford, USA, January.

Genesereth, M. R. and Ketchpel, S. P., (1994) Software Agents. *In*: Communications of the ACM, 37(7), pages 48-53.

Gibson, W., (1984) Neuromancer, Harper Collins Publishers.

Goodwin, R., (1993) Formalising Properties of Agents, Technical Report CMU-CS-93-159, School of Computer Science, Carnegie-Mellon University, Pittsburgh, USA.

Goose, S., (1995) The Design Of A Generic Customisable Distributed Communication Framework, Department of Electronics and Computer Science Technical Report 95/4, University of Southampton, UK. *Available from*: `http://www.agents.cosm.ecs.soton.ac.uk/TNG/papers/m95-4.htm`

Goose, S., (1996) Extending The Heterogeneous Communication Model: A Scalable and Seamless Architecture to Support Widely Distributed Process Interaction, Department of Electronics and Computer Science Technical Report 96/2, University of Southampton, UK. *Available from*: `http://www.agents.ecs.soton.ac.uk/TNG/papers/m96-2.htm`

Goose, S., Dale, J., Hill, G. J., De Roure, D. C. and Hall, W., (1995) Unifying Distributed Processing and Open Hypermedia through a Heterogeneous Communication Model, Department of Electronics and Computer Science Technical Report 95/6, University of Southampton, UK. *Available from*: `http://www.agents.ecs.soton.ac.uk/TNG/papers/m95-6c.htm`

Goose, S., Dale, J., Hill, G. J., De Roure, D. C. and Hall, W., (1996) An Open Framework for Integrating Widely Distributed Hypermedia Resources, to be presented at the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan, June. *Available from*: `http://www.agents.ecs.soton.ac.uk/TNG/papers/icmcs96c.htm`

Gosling, J. and McGinton, H., (1995) The Java Language Environment: A White Paper. *Available from*: `http://java.sun.com/whitePaper/java-whitepaper-1.html`

Gray, R. S., (1995) Agent TCL: A Transportable Agent System. *In*: Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management, Mayfield, J. and Finin, T., *Eds*., Baltimore, USA.

Gray, R. S., (1995b) Transportable Agents, PhD Thesis Proposal, Dartmouth College, Dartmouth, USA.

Green, C. L., Bayer, D. and Edwards, P., (1995) Towards Practical Interface Agents which Manage Internet-Based Information. *Presented at*: BCS-SGES/DTI-ISIP Intelligent Agents Workshop, Oxford Brookes University, November.

Grønbæk, K. and Trigg, R. H., (1992) Design Issues for a Dexter-Based Hypermedia System. *In*: Proceedings of the Fourth ACM Conference on Hypertext, Lucarella, D., Nanard, J., Nanard, M. and Paolini, P., *Eds*., Milan, Italy, ACM Press, pages 191-200.

Gruber, T. R., (1991) The Role of Common Ontology in Achieving Shareable, Reusable Knowledge Bases. *In*: Proceedings of Knowledge Representation and Reasoning, San Mateo, USA, Morgan-Kaufmann Publishers.

Harrison, C. G., Chess, D. M. and Kershenbaum, A., (1995) Mobile Agents: Are they a Good Idea?, IBM Research Report, IBM Research Division.

Hill, G. J., Wilkins, R. J. and Hall, W., (1993) Open and Reconfigurable Hypermedia Systems: A Filter Base Model. *In*: Hypermedia, 5(2), pages 103-118.

Jennings, N. R., (1992) On Being Responsible. *In*: Decentralised AI 3—Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds, Werner, E. and Demazeau, Y., *Eds*., Amsterdam, The Netherlands, pages 93-102.

Jennings, N. R., Varga, L. Z., Aarnts, R. P., Fuchs, J. and Skarek, P., (1993) Transforming Standalone Expert Systems into a Community of Cooperating Agents. *In*: The International Journal of Engineering Applications of Artificial Intelligence, 6(4), pages 317-331.

Johansen, D., van Renesse, R. and Schneider, F. B., (1995) Operating System Support for Mobile Agents. *In*: Proceedings of the Fifth IEEE Workshop on Hot Topics in Operating

Systems.

Johansen, D., van Renesse, R. and Schneider, F. B., (1995b) An Introduction to the TACOMA Distributed System, Computer Science Technical Report 95-23, University of Tromsø, Norway.

Kahle, B., and Medlar, A., (1991) An Information System for Corporate Users: Wide Area Information Server, Technical Report TMC-199, Thinking Machines Incorporated.

Kappe, F., Pani, G. and Schnabel, F., (1993) The Architecture of a Massively Distributed Hypermedia System. *In*: Internet Research: Electronic Networking Applications and Policy, 3(1), pages 10-24.

Kurzweil, R., (1990) Intelligent Machines, MIT Press.

Kuwabara, K., Ishida, T., and Osato, N., (1995) AgenTalk: Coordination Protocol Description for Multi-Agent Systems. *In*: Proceedings of the First International Conference on Multi-Agent Systems, page 455.

Lang, K., (1995) NewsWeeder: Learning to Filter Netnews. *In*: Proceedings of the Twelfth International Machine Learning Conference, San Francisco, USA, Morgan-Kaufmann Publishers, pages 331-339.

Levy, A. Y., Sagvi, Y. and Srivastava, D., (1994) Towards Efficient Information Gathering Agents. *In*: Software Agents—Papers from the 1994 Spring Symposium, Technical Report SS-94-03, Etzioni, O., *Ed*., pages 64-70.

Lingnau, A., Drobnik, O. and Dömel, P., (1995) A HTTP-Based Infrastructure for Mobile Agents. *In*: Proceedings of the Fourth International World Wide Web Conference, Boston, USA, pages 461-471.

Maes, P., (1994) Social Interface Agents: Acquiring Competence by Learning from Users and Other Agents. *In*: Software Agents—Papers from the 1994 Spring Symposium, Technical Report SS-94-03, Etzioni, O., *Ed*., pages 71-78.

Maes, P., (1994b) Agents that Reduce Work and Information Overload. *In*: Communications of the ACM, 37(7), pages 31-40.

Mayfield, J., Labrou, Y. and Finin, T., (1995) Desiderata for Agent Communication Languages. *In*: Proceedings of the AAAI Symposium on Information Gathering from Heterogeneous, Distributed Environments, AAAI-95 Spring Symposium, Stanford University, Stanford, USA, pages 27-29, March.

McCarthy, J, (1978) Ascribing Mental Qualities to Machines, Artificial Intelligence Laboratory Technical Report, Stanford University, Stanford, USA.

Müller, J. P., Pischel, M. and Thiel, M., (1995) Modelling Reactive Behaviour in Vertically Layered Agent Architectures. *In*: Intelligent Agents: Theories, Architectures and Languages, Wooldridge, M. and Jennings, N. R., *Eds*., Lecture Notes in Artificial Intelligence, Volume 890, Springer-Verlag, pages 261-276.

Newell, A. and Simon, H. A., (1976) Computer Science as Empirical Enquiry. *In*: Communications of the ACM, 19, pages 113-126.

Object Management Group (1993), The Common Object Request Broker: Architecture and Specification, OMG Technical Document, Number 93.12.43, Revision 1.2, December.

Open Software Foundation (1992), Distributed Computing Environment Overview. *Available from*: `http://www.osf.org/comm/lit/OSF-DCE-PD-1090-4.html`

Oren, T., (1987) The Architecture of Static Hypertexts. *In*: Proceedings of Hypertext'87, Association for Computing Machinery 1989, pages 291-306.

Ousterhout, J. K., (1994) TCL and the TK Toolkit, Addison-Wesley Publishing.

Ousterhout, J. K., (1995) Scripts and Agents, the Software High Ground. *Invited talk at*: USENIX Conference, New Orleans, USA.

Owl International Incorporated, (1987) Guide: Hypertext for the PC.

Patil, R., Fikes, R. E., Patel-Schneider, P., McKay, D., Finin, T., Gruber, T. and Neches, R., (1992) The DARPA Knowledge Sharing Effort: Progress Report. *In*: Proceedings of the Third

International Conference on Principles of Knowledge Representation and Reasoning, San Francisco, USA, Morgan-Kaufmann Publishers, November.

Pearl, A., (1989) Sun's Link Service: A Protocol for Open Linking. *In*: Hypertext'89 Proceedings, Pittsburgh, USA, ACM Press, pages 137-146.

Postel, J. and Reynolds, J., (1985) The File Transfer Protocol (FTP), Request for Comments 959, Ohio State University. *Available from*: `http://www.cis.ohio-state.edu/htbin/rfc/rfc959.html`

Queinnec, C., (1995) DMeroon: Overview of a Distributed Class-Based Causally-Coherent Data Model. *In*: Parallel Symbolic Languages and Systems '95, Ito, T., Halstead, R., and Queinnec, C., *Eds*., Beaune, France, October.

Renaud, P. E., (1993) Introduction To Client/Server Systems: A Practical Guide For Systems Professionals, Wiley & Sons.

Rosenschien, J. S. and Genesereth, M. R., (1985) Deals Among Rational Agents. *In*: Proceedings of the Ninth Joint Conference on Artificial Intelligence, Los Angeles, USA, pages 91-99.

Satoshi, H., (1996) HORB: A Distributed and Parallel Java—User's Guide. *Available from*: `http://ring.etl.go.jp/openlab/horb/doc/guide/guide.htm`

Shardlow, N., (1990) Action and Agency in Cognitive Science, MPhil Thesis, Department of Psychology, University of Manchester, Manchester, UK.

Shoham, Y., (1993) Agent-Oriented Programming. *In*: Artificial Intelligence, 60(1), pages 51-92.

Sheth, B. D., (1994) A Learning Approach to Personalised Information Filtering, Master's Thesis, Department of Electrical Engineering and Computer Science, MIT, Boston, USA.

Stamos, J. and Gifford, G., (1990) Remote Evaluation. *In*: ACM Transactions on Programming Languages and Systems, 12(4), pages 537-565, October.

Steeb, R., Cammarata, S., Hayes-Roth, F. A., Thorndyke, P. W. and Wesson, R. B., (1988) Distributed Intelligence for Air Fleet Control. *In*: Readings in Distributed Artificial Intelligence, Bond, A. H. and Gasser, L., *Eds*., Morgan-Kaufmann Publishers, pages 90-101.

Stephenson, N., (1992) Snow Crash, Penguin Books Limited.

Stoyenko, A. D., (1994) SUPRA-RPC: SUbprogram PaRAmeters in Remote Procedure Calls. *In*: Software Practice and Experience, 24(1), pages 27-49, January.

van Renesse, R., Birman, K. P., Friedman, R., Hayden, M., and Karr, D. A., (1995) A Framework for Protocol Composition in Horus. *In*: Proceedings of Principles of Distributed Computing, August.

Vere, S. and Bickmore, T., (1990) A Basic Agent. *In*: Computational Intelligence, 6, pages 41-60.

Voorhees, E. M., (1994) Software Agents for Information Retrieval. *In*: Software Agents—Papers from the 1994 Spring Symposium, Technical Report SS-94-03, Etzioni, O., *Ed*., pages 126-129.

Weihmayer, R. and Veltuijsen, H., (1994) Applications of Distributed Artificial Intelligence and Cooperative Problem Solving to Telecommunications. *In*: AI Approaches to Telecommunications and Network Management, Leibowitz, J. and Prereau, D., *Eds*., IOS Press.

White, J. E., (1994) Telescript Technology: The Foundation for the Electronic Marketplace, General Magic White Paper GM-M-TSWP1-1293-V1, General Magic Incorporated.

White, J. E., (1994b) Mobile Agents make a Network an Open Platform for Third-Party Developers. *In*: Computer (IEEE Computer Society), 27(11), pages 89-90, November.

White, J. E., (1995) Telescript Technology: An Introduction to the Telescript Language, General Magic White Paper GM-M-TSWP3-0495-V1, General Magic Incorporated.

Wilkins, R. J., (1994) The Advisor Agent: A Model for the Dynamic Integration of Navigation Information within an Open Hypermedia System, PhD Thesis, University of

Southampton, Southampton, UK.

Wooldridge, M. and Jennings, N. R., (1995) Intelligent Agents: Theory and Practice. *In*: Knowledge Engineering Review, 10(2), June.